NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CLOUD COMPUTING IN SUPPORT OF SYNCHRONIZED DISASTER RESPONSE OPERATIONS

by

Shawn M. Kelly
Corey A. Mazyck

September 2010

Thesis Co-Advisors: Man-Tak Shing
Karl Pfeifer

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2010	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Cloud Computing in Support of Synchronized Disaster Response Operations		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Shawn Kelly and Corey Mazyck			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number _____N.A._____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE** A	

13. ABSTRACT (maximum 200 words)

During disaster response, key resources are supplied from a variety of channels including: government agencies, volunteer organizations, commercial businesses, educational institutions and others. While many of the entities have efficient internal methods of communication and coordination, global collaboration has historically been hindered by political, social, and technological challenges. Following Hurricane Katrina this resulted in over-resourcing of some in-need areas with little or no resources reaching others. While there is little argument that a global approach to disaster response should be adopted, political and technical challenges surrounding the integration and ownership of such a system have prevented its emergence.

This thesis examines the current challenges to collaboration between responding entities and proposes self-synchronization using a distributed, highly scalable, Web application based on cloud computing technologies to facilitate communication between a broad range of public and private entities without requiring them to compromise security or competitive advantage. The proposed design applies the unique benefits of cloud computing architectures such as automatic scaling, geographic distribution, and query performance to the disaster response domain.

14. SUBJECT TERMS Cloud Computing, Synchronization, Collaboration, Elasticity, Disaster, Synchronized, Common Operating Picture, Framework			**15. NUMBER OF PAGES** 126
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

CLOUD COMPUTING IN SUPPORT OF SYNCHRONIZED DISASTER RESPONSE OPERATIONS

Shawn M. Kelly
Major, United States Marine Corps
B.A., California State University, Long Beach, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT
and

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

Corey A. Mazyck
Major, United States Marine Corps
B.A., Campbell University, 2003

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2010

Author: Shawn M. Kelly
Corey A. Mazyck

Approved by: Man-Tak Shing
Thesis Co-Advisor

Karl Pfeiffer
Co-Advisor

Peter Denning
Chairman, Department of Computer Science

Dan Boger
Chairman, Information Sciences Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

During disaster response, key resources are supplied from a variety of channels including: government agencies, volunteer organizations, commercial businesses, educational institutions and others. While many of the entities have efficient internal methods of communication and coordination, global collaboration has historically been hindered by political, social, and technological challenges. Following Hurricane Katrina this resulted in over-resourcing of some in-need areas with little or no resources reaching others. While there is little argument that a global approach to disaster response should be adopted, political and technical challenges surrounding the integration and ownership of such a system have prevented its emergence.

This thesis examines the current challenges to collaboration between responding entities and proposes self-synchronization using a distributed, highly scalable, Web application based on cloud computing technologies to facilitate communication between a broad range of public and private entities without requiring them to compromise security or competitive advantage. The proposed design applies the unique benefits of cloud computing architectures such as automatic scaling, geographic distribution, and query performance to the disaster response domain.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ALAN	American Logistics Aid Network
API	Application Programming Interface
COTS	Commercial Off-The-Shelf
FEMA	Federal Emergency Management Agency
GIS	Geographic Information System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IP	Internet Protocol
ISO	International Organization For Standardization
IT	Information Technology
KML	Keyhole Markup Language
LAT-LONG	Latitude-Longitude
LOGCOP	Logistics Common Operating Picture
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
PaaS	Platform as a Service
SaaS	Software as a Service
SDR-1	Synchronized Disaster Response Version 1
SOA	Service Oriented Architecture
URL	Uniform Resource Locator
USNORTHCOM	United States Northern Command
W3C	World Wide Web Consortium
XML	Extensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

We would like to extend our utmost gratitude to Dr. Man-Tak Shing and Dr. Karl Pfeiffer for their mentorship, continued guidance, and unending patience over the past year. We would also like to thank the U.S. Northern Command, namely Roger Smith, for bringing this project to us and continuing to remain engaged until the end.

Shawn Kelly

I extend most sincere appreciation to my wife, Sophia, my boys, Drake and Caleb, and to my parents, Pat and Mike. Thank you for your continued love, support, and guidance. Without you, this achievement would not be possible. It was your commitment and understanding that enabled me to devote so much to this project. There were multiple times during the last two years that my focus was away from family. You have endured all of this and for that I express my tremendous gratitude.

Corey Mazyck

I, first and foremost, give all glory and honor to my Lord and Savior Jesus Christ who orders my every footstep. I can do nothing without you. To my wife, Pam, I truly am grateful for all of your continued love and support. You are the wind beneath my wings. To my beautiful daughters, Courtnee and Kayla, daddy loves you very much and understands that these personal sacrifices were not mine alone. I thank you for your patience and tolerance over the past two years.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

During disaster response, key resources are supplied from a variety of channels including government agencies, volunteer organizations, commercial businesses, educational institutions and others. While many of the entities have efficient internal methods of communication and coordination, global collaboration has historically been hindered by organizational and technological challenges. Following Hurricane Katrina, this resulted in over-resourcing some in-need areas with little or no resources reaching others. While there is little argument that a global approach to disaster response should be adopted, challenges surrounding the integration and ownership of such a system have prevented its emergence.

B. DISASTER RESPONSE

Disasters, natural or man-made, can be gradual or sudden and include events such as earthquakes, tsunamis, and cyber attacks. They may also be cyclical as in the cases of hurricanes and snowstorms, which have their own season (Wassenhove, 2006).

Each year, approximately 200 million people are impacted and 150,000 deaths worldwide are attributed to crises and disasters. Natural disasters alone displace over 5 million people annually (Thomas, n.d.). On the relief frontline, responders often find themselves challenged by hostile environments while attempting to assist displaced victims.

There are four phases of a disaster relief effort: preparedness, response, recovery, and mitigation. This thesis focuses on the first two. During the preparedness phase, an assessment must be completed in order to get an understanding of the potential disaster and to pre-stage the anticipated resource requirements. During the response phase, efforts are taken to relieve individuals affected by a disaster. These efforts include

providing lifesaving resources, conducting search and rescue operations, providing medical assistance, and in some cases, deterring crime.

Disaster relief efforts require that response teams work together in a cohesive manner before, during, and after an operation. The first 48 to 72 hours are the most critical. During this period, when both survivors and responders are likely to be disoriented, response efforts are less effective. A considerable number of lives will be saved or lost depending on how well organizations are prepared and how effectively they coordinate response efforts.

The primary responsibility for coordinating disaster response in the United States belongs to the Federal Emergency Management Agency (FEMA), which falls under the Department of Homeland Security. As part of the Agency's role in disaster preparedness, FEMA has recognized resource distribution as a key element of the initial disaster response and has identified eight resources critical to the response including: water, ice, meals, generators, blankets, sheeting, tarps, and cots. Participation from a wide range of public and private entities is crucial to successful distribution. Synchronization between these entities is paramount (Fenton, 2010).

C. CHALLENGES TO DISASTER RESPONSE

1. Disasters Are Hard to Predict

A primary component of disaster response is logistics. In normal environments, this is predictable and allows organizations to react quickly to surging demand. Logisticians can select the most efficient transportation routes based on well thought out metrics and algorithms. Supply chains can be optimized over time to ensure products are assembled and delivered just as they are needed, maximizing profit and limiting overhead. Like commercial logistics, responders' primary objective is getting the right resource to the right place at the right time. Unfortunately, the place and time are not given in advance so setting up and testing efficient logistics supply chains is extremely difficult (Wassenhove, 2006).

Leadership and communication are other important components to disaster response. Local governments may be stressed and communication systems may be unusable. Responders cannot necessarily depend on establishing communication or receiving direction from outside sources. Internal processes are likely to be disrupted as well.

2. Disaster Response Is Hard to Coordinate

a. Top-Down Approach

Assuming leadership and communication infrastructure remain intact, the top-down approach to directing disaster response is still inherently challenging due to the responding number of organizations. Even when jurisdictional and organizational barriers are eliminated, synchronizing the actions of so many entities takes time. Information coming from various sources must flow up to the lead organization, which attempts to match capabilities with needs before directing a response. The time consumed through this process translates to a delay in the initial response. With loss of leadership and communication infrastructure, the delay increases.

b. Extraordinary Circumstances

Collaboration between diverse organizations, which is difficult under normal conditions, can be nearly impossible in the chaotic environment resulting from a disaster. As pointed out by Heide (1989), there are several factors that impede teamwork in the aftermath of a disaster.

> One of the reasons disaster response is difficult to coordinate is because disasters are different from routine, daily emergencies. The difference is more than just one of magnitude. Disasters generally cannot be adequately managed merely by mobilizing more personnel and material. Disasters may cross jurisdictional boundaries, create the need to undertake unfamiliar tasks, change the structure of responding organizations, trigger the mobilization of participants that do not ordinarily respond to local emergency incidents, and disable the routine equipment and facilities for emergency response.

Heide's statement focuses primarily on governmental response at the local, state, and national levels, but commercial organizations have significant resources to contribute as well. Incorporating them into the response efforts adds another layer of complexity to the already challenging collaboration problem.

c. People and Organizations

Many of the problems coordinating disaster relief efforts can be attributed to individuals themselves, which are unwilling or unable to adapt to the dynamic disaster environment. Organizations are inclined to center on core competencies rather than looking externally to coordinate efforts. When external coordination is sought, it is more likely to occur between organizations that are familiar and comfortable with one another.

This is not just a problem between organizations in different industries but within industries as well. In order to survive, commercial organizations must gain and protect competitive advantage. Sharing detailed information on capabilities and resources could jeopardize their place in the market. Public entities also have a responsibility to safeguard information and protect the knowledge that describes their structure and internal processes.

Coordination is further complicated by incompatible rules and protocols governing the various organizations. One would think that bureaucracy would not play much of a role during disaster relief efforts, but politics and jurisdictional authority often stifle coordination by injecting additional barriers (Heide, 1989).

d. Interoperability

A lack of technical interoperability between organizations contributes to teamwork shortfalls. Public organizations are responsible for disaster response systems tailored to particular situations and budgets, while commercial organizations streamline their systems to their respective markets. During a disaster, these public and private entities' efforts to work together are hindered by technical barriers that prevent information sharing with two main problems: the data itself and the protocols to share it. Information about similar resources is often described and stored in different ways by

different organizations, which makes data from one organization unusable to another. The systems and their protocols are often proprietary and incompatible, compounding the problem. The result is a stove-piped approach to disaster response in which each organization only sees the information it controls.

3. Network Architecture Problems

The client-server architecture is prevalent in the disaster response community today but cannot fully address the challenges likely to arise from significant disasters. While there are many benefits to the client-server approach, implementation often falls short in the areas of scalability and robustness, two critical attributes for disaster response systems.

Normal usage for disaster response systems is relatively low, requiring only a few servers. When disasters strike, usage surges; this requires many servers to handle the increased load. If a sufficient number of servers aren't available, performance will degrade or the system will crash. The common solution is to add more servers with the ultimate number being driven by budget, but it is sometimes difficult to justify dozens of servers running idly in case they are needed for a disaster.

The physical placement of servers is also important. Servers are often located within the buildings and geographic areas they are intended to support. This is fine for smaller events but presents a problem when infrastructure is destroyed. Emergency generators and redundant servers are not sufficient in these situations. The only true measure of protection is to distribute servers geographically.

These two problems have the potential to compound one another during a disaster, and the responders have to overcome damaged systems to do their jobs.

D. MEETING THE CHALLENGES

The disaster relief community consists of various systems. Together, these systems are able to perform functions such as finance accountability, tracking response efforts, and managing and outputting reports. However, there still continues to be an

automated situational awareness void. No system exists to help responding entities cooperatively to deliver resources to the right place at the right time. Technology is no longer the limiting factor (Thomas, n.d.). The challenge is finding a way to allow diverse entities to maintain varying organizational structures and systems, while allowing them to share a single view of the evolving situation. A concept referred to as self-synchronization can provide such a framework for sharing information required for an effective, coordinated disaster response.

E. SELF-SYNCHRONIZATION

When each component or individual in a system works separately and without external direction, using shared information to accomplish a common objective, the system is said to be self-synchronized. This decentralized approach allows organizations to base their actions on a shared worldview without requiring them to coordinate directly or wait for direction. The concept is well suited to the disaster response domain, which requires participation from a wide range of public and private organizations to be effective. A system developed around this concept could provide a neutral zone for organizations to share information describing resource distribution and requirements, allowing each entity to maintain situational awareness while remaining independent and agile. The result would be a rapid event and systematic distribution of resources.

Understandably, some organizations may be resistant to information sharing for a number of reasons. Commercial organizations want to maintain their competitive advantage and public organizations need to maintain a level of security or classification around information they control. With this understanding, information needs to be shared in abstract terms to keep the barriers to entry low and encourage voluntary participation.

F. FOCUS OF THIS THESIS

This thesis will focus on developing a system to facilitate information sharing across a diverse range of responding organizations in the disaster response domain. It will be guided by the following questions:

1. What are the technical and organizational challenges associated with the coordinated allocation of resources in disaster response situations?
2. Which technological approaches best address those challenges?
3. What are the software and hardware requirements for a system that would address those challenges?
4. What software system design and implementation, based on the aforementioned requirements, would best address challenges associated with coordinated allocation of resources in disaster response situations?

G. APPROACH

Authors met with members of United States Northern Command (USNORTHCOM) to better understand the current challenges and potential solutions to collaboration in response to a disaster. Research was conducted on existing commercial off-the-shelf solutions and implementations including USNORTHCOM's concept for a Logistical Common Operating Picture (LOGCOP). There was also a thorough review of network applications, Web applications, and Internet architectures to see which applications and architectures best suit the disaster response domain.

A set of features based on input from USNORTHCOM was developed and prioritized as critical, important, or useful. Due to time constraints, only critical features are included in the initial design of the system. Functional and nonfunctional requirements were derived from the feature set and better understood through use cases. From these artifacts, a Web application based on Google's App Engine platform was designed. The high-level design was sent to USNORTHCOM for review with their feedback serving as the basis for future work.

H. ORGANIZATION OF THESIS

The remainder of this thesis is organized as follows:

Chapter II provides the background and research for this thesis including a review of current software solutions and multiple network architectures. Cloud computing is

examined as a new architecture with potential to address problems inherent to the disaster response domain. Google's App Engine is examined in-depth as a potential cloud based platform on which to develop a solution.

Chapter III describes the functional and non-functional requirements for the proposed disaster response system based on information provided by USNORTHCOM and research captured in Chapter II.

Chapter IV proposes Cloud Computing as a technical solution and describes a high-level design based on the Google App Engine platform to satisfy the requirements established in Chapter III.

Chapter V summarizes the thesis, identifies its key contributions, and outlines a framework for future work based on feedback from USNORTHCOM.

II. BACKGROUND AND LITERATURE REVIEW

In the following section, some commercial off the shelf logistics systems and some non-commercial systems that are being used in support of disaster relief are reviewed. This thesis examines existing information technologies that can potentially support the implementation of a software solution to facilitate information sharing among disaster relief organizations.

A. CURRENT SOFTWARE

1. Commercial Off-the-Shelf Software Systems

a. SCB Software

SCB software is an emergency management database that maintains information about current agencies which provide emergency services. It stores information on resources, resource centers, communications, tasks, and shelters. The system uses charts to facilitate response operations by tracking the incidents and resource distributions. It further enables users the ability to track resources, allowing them to be located quickly following a disaster. This system falls short in that it does not allow responders to track logistics resources that are in transit. This is important to prevent resource saturation in certain areas. In transit distribution is also significant to synchronize resource distribution (SCB Software Web site, n.d.).

b. InMotion Global Home Logistics, Lean Logistics, Appian Logistics Software, Inc.

InMotion Global Home Logistics, Lean Logistics, and Appian Logistics Software focus on just-in-time and third-party logistics, which are important in commercial sectors. These popular systems provide transportation management software support, allowing organizations to effectively choose routes that increase transportation efficiency. Some provide real-time data via Web-portals and can be adjusted to fit an

organization's specific needs. Unfortunately, none of these systems significantly contribute to disaster response since they require more predictability than is possible in this domain. They were made to increase the efficiency in commercial transportation but not in relief efforts. They are best suited for their primary purpose.

c. LogiMax

LogiMax "is a browser-based system that contains everything public warehouses need to be successful" (LogiMax Web site, n.d.). Its main purpose is to optimize warehouse inventories. Efficient warehouse management saves companies money. Warehouse management software such as this is currently being used in conjunction with other disaster relief software to manage warehouse inventories within organizations during disaster relief operations. They do not, however, solve the problem of interoperability between organizations.

2. Disaster Relief Systems

a. Web Sites

Numerous relief organizations have Web sites, which solicit monetary donations. These include: National Voluntary Organizations Active in Disaster, Direct Relief International, Clinton Bush Haiti Fund, UNICEF, and World Vision. While they are instrumental in collecting money for relief efforts, they do not provide significant support for the initial disaster response.

b. Web-Portal

The American Logistics Aid Network (ALAN) was created after Hurricane Katrina. This organization provides a Web-portal for commercial and non-profit organizations that have decided to work together by donating money and resources under a common umbrella for future disaster relief efforts. The ALAN organization in turn gives media attention to its donor organizations. The ALAN Web-portal displays the real-time resource needs that the organization requires to continue operating and is

organized by category (American Logistics Aid Network Web-portal, n.d.). The Web-portal is very simple to understand, easy to use, and should work to assist in preparing for future disaster relief efforts. The advantage to this system is that it encourages businesses and donors to work together, and it provides an intuitive interface to assist them. It does not, however, facilitate collaboration between disaster responders to provide effective resource distribution.

c. Geospatial Information System (GIS)

A geospatial information system (GIS) uses hardware and software to represent different aspects of geography, allowing users to input and query data from different viewpoints. GIS preserves data, allows for data sharing, and increases decision-making. This technology can be used to provide enhanced and timely data with a cost effective approach. Numerous industries, non-governmental organizations and state governments use this technology (What is GIS, n.d.). GIS systems are generally used to identify environmental conditions but could probably be adapted to provide some level of logistical support. These technologies should be incorporated into the proposed system but are not a complete solution.

B. USNORTHCOM LOGISTICS COMMON OPERATING PICTURE

1. Current System

The United States Northern Command (USNORTHCOM) Logistics Common Operating Picture (LOGCOP) is a user defined operational picture that has numerous layers overlaid on Google Earth imagery. It consists of over 250 data layers of critical infrastructure, other infrastructures, geological and event-driven data. It is comprised of over 30,000 private sector data points.

A disadvantage to USNORTHCOM's approach is that its LOGCOP efforts are labor intensive and absent of automation. The LOGCOP is a compilation of several databases and ad hoc programs that are not interoperable and require human interaction

to gather, transfer, and convert the data between the systems. In order for information to be relevant, information must be updated manually in real-time or near real-time, requiring additional work.

2. Future System Features

USNORTHCOM visualizes a system that would allow governments, commercial businesses, non-governmental organizations, faith-based groups, and volunteer organizations to conduct theater logistics collaboratively by sharing information through a common operating picture. This addition would empower synchronizing organizations thus entities would no longer have to use a top-down leadership approach for resource distribution.

The goal is to provide a central, single source information sharing system that responders can use to communicate details about the resource situation before committing resources. The system would be populated at local, state, regional, and federal levels. A single source information system would eliminate individual organizations from having to rely on disparate systems.

Table 1 presents a summary of the features of the aforementioned software systems.

	Automated	Logistics Resource Status	Warehousing	Transportation	Information Sharing	Locating Resource	Solicit Money
SCB Software	✓					✓	
InMotion, et al.	✓			✓			
LogiMax	✓		✓				
Web sites	✓						✓
Web-portal	✓				✓	✓	✓
GIS	✓	✓			✓	✓	
LOGCOP		✓			✓	✓	

Table 1. Software Technologies

C. ARCHITECTURES FOR NETWORK SYSTEMS

In this section, we review existing architectures and Web 2.0 technologies to identify, which is best suited to Web application development for the disaster response domain.

1. Client-Server Architecture

Client-server architectures distribute applications between client and server systems which communicate with each other over a network. A client sends requests over a network to a server, which performs necessary operations and responds to the client.

There are several benefits to the client-server model. Interfaces are typically designed for the lowest common denominator with regard to network bandwidth, which is potentially useful when relying on cellular, satellite, or hastily-formed networks. The application itself is deployed from a single location, which makes updating it much easier. The client-server architecture allows flexibility with regard to the capabilities of the clients. If a client's system is limited, it can take advantage of the server's processing speed and storage capabilities: faster servers result in faster responses. There is a downside. Since a server is typically responsible for handling hundreds of simultaneous requests, when one application server is not sufficient, either a cluster of servers has to be implemented to handle incoming traffic or more data services must be added to support growing sets of data. Unfortunately, adding more hardware cannot usually be accomplished quickly or inexpensively, which presents a problem in the disaster response domain. There is also a downside to having servers centrally located as they constitute a single point of failure if the data center is destroyed or disabled during the disaster.

2. Web Services

Web services are software systems designed to provide machine-to-machine interoperability across a network by defining a common interface and data format (Booth et al., 2004). Most commonly, information is formatted as Extensible Markup Language

(XML) messages that follow established standards such as Simple Object Access Protocol (SOAP) or Representational State Transfer (REST). Systems that use Web services are divided into two types: consumers and providers. Neither relies on direct human interaction. Web services provide a potential solution to the interoperability problem that exists between disaster response systems, but implementing them effectively requires significant efforts and costs on the part of responding organizations. Security concerns also raise the barriers to entry of such a system. It is unrealistic to assume that many organizations would invest the time to develop interfaces required to consume these services even if they were available and secure.

3. Service Oriented Architecture

A Service Oriented Architecture (SOA) is an architecture in which software components are provided as functional Web services over the Internet. These components are clusters of Web services designed to work together as a complete system. By combining components, SOAs provide much more capability than individual Web services alone. Like Web services, they are designed to provide machine-to-machine interactions using standard interfaces but with the added abilities for systems to search and find the services they need in a directory and then bind them automatically. This approach may be feasible for public organizations where a single framework can be directed and implemented from the top down, but it is an unrealistic way to bring public and private entities together. Even if an open framework were developed, SOAs have the same barriers to entry as Web services.

D. CLOUD COMPUTING

1. Cloud Computing in General

The term "cloud" is often used when something is understood at a high level, but the details about how it works are hidden. This concept is referred to as "abstraction" since something complex is described in very simple and abstract terms. The Internet itself is usually represented as a cloud because details describing the links, routers,

switches, servers, and other infrastructure are hidden (Chu-Carroll, 2010). Often, users do not know what computers they are ultimately connecting to when they type the uniform resource locator (URL) addresses into their browsers. More importantly, they do not care as long as the services provided meet their needs. Cloud computing embraces this concept by providing resources as services over a network in the form of computing infrastructure, development platforms and software that allow consumers to concentrate only on things that are important to them.

The Internet is comprised of millions of computers sitting in data centers around the world. Traditionally, a particular Web application will run continuously on one or more servers, waiting to handle incoming requests. When incoming traffic surges, the application is limited by the capabilities of the server or servers it is running on. When incoming traffic stops, the application becomes idle. This approach is insufficient during high demand and inefficient during low demand.

Cloud computing solves these problems by providing computing and storage as services rather than offering physical machines. In the cloud, a Web application does not run continuously on dedicated machines. Instead, the application sits dormant until it is needed. At which point, an instance of the application is started on any of the thousands of available servers. As the limits of each server are reached, new instances are started on different servers to accommodate incoming requests. As demand decreases, applications terminate, freeing up the server resources. Data is handled using a similar approach with multiple copies of datasets spread across geographically distributed machines.

2. Types of Cloud Services

There are three main categories of cloud services including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Each level provides more functionality than the one before it.

Data centers require significant investment in infrastructure such as power, cooling, storage, servers, and high-speed network connections as well as the staff needed to install, test, and maintain the systems. Few companies have the financial resources or

technical expertise to build their own infrastructure. IaaS typically leverages virtualization to provide infrastructure in the form of computing power and storage as a service over the Internet. Computing cycles and storage are metered and billed to the consumers as they are used, usually on a monthly basis. Since resources are essentially rented, initial costs are low. More importantly, scaling resources to match user demand is automatic. One example of an IaaS provider is Amazon, which provides computing and storage through Amazon Elastic Compute Cloud (EC2) and Simple Storage Service (S3), respectively. EC2 users can launch and configure instances of operating systems in which they can run their applications, while S3 users have access to unlimited space for data storage (Amazon Web Services, 2010).

PaaS takes this concept to the next level by adding a stack of services to the infrastructure, providing a robust platform to facilitate development of applications or other services that can take advantage of the underlying computing power and storage provided by the infrastructure. The concept is the same as IaaS with resources metered and billed as they are used, but the target audience is comprised of developers looking to create cloud-based applications and services without having to deal with configuring operating systems or dealing with servers. They leverage application programming interfaces (APIs) and tools provided through the platform to accelerate development and deployment of their applications (Sun Microsystems, 2009). Google's App Engine is one such development platform. Developers use standard languages such as Python or Java to develop Web applications that take advantage of Google's vast computing power and storage space through APIs and services provided by Google. Once up and running, these applications reside in Google's cloud and are made available to users as SaaS.

SaaS refers to software deployed over the Internet rather than on individual machines. In most cases the software is accessed via Web browser. Consumers do not need to deal with purchasing and maintaining servers or installing and updating software. One of the biggest benefits is services are available at any location with Internet access, so users do not need to store documents on thumb-drives or to e-mail files home to continue working on them, which saves time and significantly reduces version problems. This concept becomes very powerful in collaborative applications when many users in

different geographical areas share a single document, file, or set of data. Documents and storage are backed-up by services as well, providing an additional level of protection. There are many good examples of SaaS ranging from free applications such Hotmail and Google Apps to complete business solutions such as SAP Business ByDesign.

3. Key Considerations for Cloud Computing

Availability is one key benefit to cloud computing. Since clouds consist of thousands of servers that dynamically adjust to handle changing demands, individual servers or entire data centers can be shut down without disrupting service. Resources and services can also be provided by data centers that are physically located closest to the incoming request. These abilities are extremely useful for maintaining uptime during periods of maintenance, power failure, or high Internet usage and are particularly important in domains like disaster response where responders depend on access to their respective system to act quickly and efficiently.

The distributed architecture also allows cloud computing to be disaster resistant. When a disaster occurs in one geographic location, only a fraction of the thousands of worldwide servers will be affected. Data and applications backed up in geographically distributed data centers will continue to run without interruption. This is unlike traditional client-server architectures where a single server located near the disaster area could result in total loss.

Elasticity is probably the clearest benefit to selecting cloud computing over other Web-based architectures. As discussed previously in this chapter, cloud based systems are designed to allocate computing and storage resources dynamically, balancing incoming requests across machines or data centers. This is accomplished through virtualization technologies, which allow applications to be decoupled from physical servers. Instances of many different applications could run on a single server for lightweight applications, while applications requiring more resources could be started on many servers at once. They are never installed or saved on the application servers. They run in memory as they are needed and terminate when they are not. This setup allows applications to be created with very low upfront costs, since few resources are consumed

during development and testing. More importantly, the application will scale from a few users to a few million in a matter of seconds, providing uninterrupted services to unanticipated surging demand.

There are some potential pitfalls to cloud computing that also need to be considered. The primary benefits of cloud computing, elasticity and redundancy, require that applications and data be distributed and replicated across data centers. This presents security issues, especially when those data centers reside in other countries, which have different data protection laws. There are many approaches to addressing these concerns, ranging from simple encryption to development of tightly controlled private clouds. Some entities may choose to avoid cloud computing all together for sensitive data. Whatever the approach, security needs to be considered before sensitive data is deployed and replicated across thousands of servers.

Another concern with cloud computing is data ownership. Who really owns the data once it is deployed across data centers? When a user pays for a cloud service, the user is essentially outsourcing data storage, relinquishing some amount of control to the service provider. This issue becomes more complicated when the provider of SaaS relies on a different entity to provide IaaS. In all cases, users and organizations need to thoroughly read and understand service agreements and contracts before entrusting their data to service providers. As with security, the sensitivity and value of data need to be considered. Once proprietary or private data has been compromised, damages may be irreversible.

E. GOOGLE CLOUD

There are various corporations that provide cloud computing infrastructure, platforms, or software as a service including Google, Oracle, Amazon, Microsoft, and Netsuite. Google, known for its search engine, provides software as a service in the form of Gmail, Google Calendar, Google Sites, Google Talk, Google Video, and Google Message Security. Google uses cloud computing to support a number of these applications.

1. Google Data Centers

Google's cloud consists of several state-of-the-art data centers. These massive facilities handle data crunching for Google's search engine and other Web-based applications. They also include massive amounts of distributed storage needed to hold information describing Web sites across the Internet. Every search term ever submitted to the Google search engine is stored somewhere in one of Google's data centers. Each data center is physically secure and climate controlled. And, just one data center alone can contain up to 45,000 servers (Google Data Center Video, 2010).

2. Bigtable

Bigtable is the base for data storage at Google. It is not a database but a distributed, persistent, multi-dimensional sorted map, which manages petabytes of data across thousands of servers. Even though Bigtable is similar to a database, it does not use the relational database model. Google clients' store their data on Bigtable where their data can be spread across various servers. Google uses this system to store a number of its projects, including Google Web Indexing and Google Earth. Google applications place varying demands and latency requirements on Bigtable; its flexibility allows it to provide high performance solutions to meet those demands. It is a self-managing system that will automatically adjust when servers are added or removed. "Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. It is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth" (Chang, et al., 2006).

3. Google App Engine

App Engine is provided by Google as a PaaS for developers that want to take advantage of Google computing and data resources to provide their own SaaS. Google App Engine lets users build high-traffic Web applications without having to manage high-traffic infrastructure" (About Google App, n.d.). A user can build an application on a workstation or laptop using one of Google App Engine's primary programming

languages and subsequently deploy it on Google's Cloud. Like all PaaS, developers are only billed for the resources consumed each month with the first 500 megabytes of storage and 5 million page-views free of charge. Users only pay for resources used in excess of their free monthly resource allocation. And as a developer, one can limit or control the maximum number of resources an application uses. This feature protects the developer from over spending. The platform can also be used to give access to some users while denying privileges to others (Chu-Carroll, 2010). Google App Engine uses a system called Datastore as a scalable alternative to a database for persisting data. Detailed information on Google App Engine and the Datastore is available in Chapter IV and Appendix B.

III. REQUIREMENTS

A. INTRODUCTION

1. Purpose

The proposed system, Synchronized Disaster Response version 1.0 (SDR-1), will be a publically available system designed to foster real-time information sharing openly and transparently across all levels of the public and private sectors, resulting in efficient use of resources during national disasters. The system will be able to receive and maintain information, including location, quantity, and description of critical resources. The information will be attained from public, private, educational, volunteer, and faith-based organizations. Once a disaster occurs, the system will be able to efficiently generate a common picture to show critical resource shortfalls and provide detailed information to government, commercial and volunteer entities to improve joint response. Organizations moving critical resources will also be able to post resource data, which will include the type and quantity of resources being transported, the intended location, the final destination, and the time of arrival. The SDR-1 system will include easy to use Internet-based forms for data entry, high-speed data storage, a graphical map-based interface to display the location of resources needed, resources in transit, and resources distributed. This chapter describes the functional and non-functional requirements for SDR-1.

2. Assumptions and Dependencies

The success of SDR-1 will depend on several factors, including participation from a diverse group of public and private entities, the reliability and availability of wide area networks in areas from which response is being coordinated, and the accuracy of information submitted by users with those entities.

B. SYSTEM OVERVIEW

1. Stakeholders

a. Organizations

SDR-1 will provide collaboration across a diverse set of private and public entities and users in response to a significant event. Entities can be divided into five general categories: government, commercial, non-profit, faith-based, and volunteer. Each entity is comprised of one or more potential users, which can be categorized into one of three general groups: managers, responders, and administrators.

(1) Government entities such as USNORTHCOM, Federal Emergency Management Agency (FEMA), National Guard, and fire and police need a common operating picture describing the quantities and locations of specific resources as well as the ability to identify locations where resources are needed for the purpose of disaster response planning. USNORTHCOM is looking for a solution to share information with other public entities as well as build awareness of resources potentially available from private entities.

(2) Commercial entities such as Wal-Mart, Home Depot, or small privately owned stores are interested in supporting their respective communities in preparation for or in response to disasters. In order to accomplish this, they need an easy way to share information describing critical resources contained at each store location. Many private entities will volunteer to transport resources directly. These entities need access to shared information to see where their contributions could best be used.

(3) Non-Profit Organizations, such as the American Red Cross, play a significant part in the collection and distribution of critical resources. They need to share information describing the status of their resources and established distribution locations with public agencies. They also want to build situational awareness through a common operating picture.

(4) Volunteer organizations and faith based groups want information about how they can best contribute to disaster response. Groups with the means to collect and distribute critical resources want a means to gather situational awareness without hindering professional responders. They need an easy method for discovering where resources might be needed as well as an easy method to communicate where they have distributed resources. Additionally, they will provide information to the system describing resource shortfalls they will discover.

b. Users

SDR-1 must support a variety of users who will access the system to contribute, view, or extract data. The users can be divided into various categories:

(1) Responders are personnel who actively provide relief assistance during a disaster response. Some examples of responders are helicopter crewmen, police officers, fire and rescue support, relief organizations, and religious groups. Responders also include operations personnel or other liaisons that provide data between the system and other responders.

(2) Managers are those individuals within organizations who have direct knowledge of the organizations' resources and are authorized to submit that information to SDR-1. They have the wherewithal to forecast the quantities and status of resources on hand, and they manage their organizations' resource data in SDR-1.

(3) Administrators are those select individuals who have additional privileges in SDR-1. They are charged with a wide range of duties required to maintain the system, identify and correct problems, and assist other users as necessary.

2. Primary Stakeholder Needs

a. Overview

While the system will provide collaboration across a diverse group of stakeholders, every aspect of the system can ultimately be traced back to the specific

needs of the users. In order to foster improved collaboration between the various entities involved in disaster response, the following needs must be addressed:

(1) Users need the ability to collect, maintain, and distribute information describing the location and quantity of critical resources from a wide range of public and private entities.

(2) Users need the ability to collect information describing resources needed at a specific location, resources in route to a specific location, and resources that have arrived at a specific location from a wide range of public and private entities.

(3) Users need the ability to collaborate with public and private entities using a logistical common picture to share detailed information about resources in a geographic area, including available resources and requests for resources, across all levels of the public and private sectors using a Web-based mapping solution.

(4) Users need a robust and highly available system with the ability to automatically scale in response to increased demand.

(5) Users need a highly intuitive system that can be learned in a few hours.

3. System Features

Stakeholder needs can be satisfied by the set of features below, which address both function and quality aspects of the system. The features have been divided into three priority levels: critical features, important features, and useful features. The baseline for SDR-1 delineates the features that will exist in the first developmental iteration.

Critical Features

F01: User account creation and management

F02: Organization creation and management

F03: Create stores / distribution locations within an organization

F04: Describe the resource situation at various geographical locations, including the need for resources, the delivery of resources, and the

resources in transit

F05: Query and filter data, returns a data set

F06: Export query results as a KML file

F07: Scalable to thousands of simultaneous users

F08: Disaster Resistant: system must be able to survive a disaster

F09: Remain operable in areas with low bandwidth

----------Baseline for SDR-1 v1.0----------

Important Features

F10: Describing resources for multiple distribution locations

F11: Display search results for organizations or resources on a virtual map

F12: Performance: Real-time update of information

F13: Reliability near 100%

F14: Availability near 100%

F15: Intuitive Web-based user interface

Useful Features

F16: Based on open-source programming language and common practices

4. Specific Functionality

a. Overview

The functional aspects of SDR-1 can be divided into four high level functional requirements: User Management, Organizational Management, Resource Information, and Common Operating Picture. High-level requirements can be summarized by the use case diagram in Figure 1.

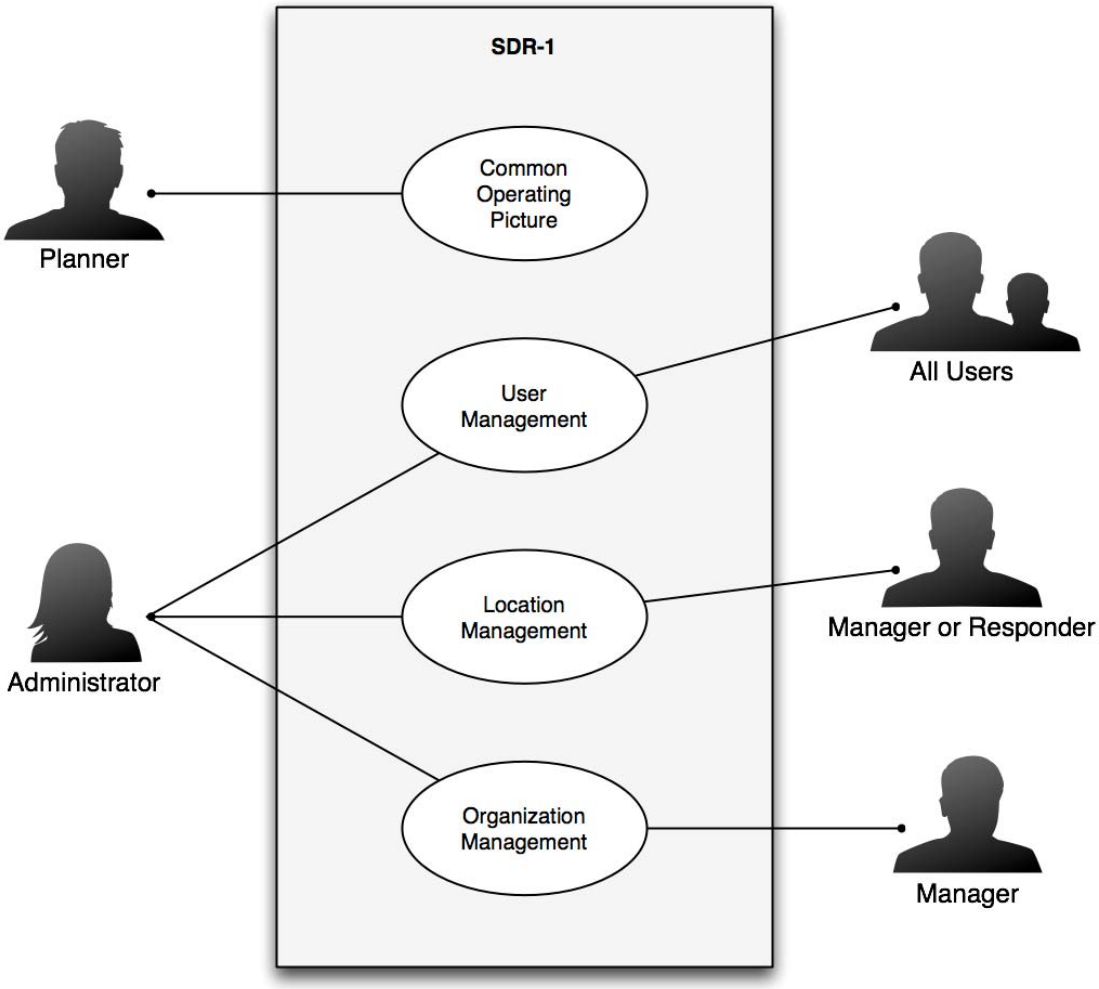


Figure 1. High-Level Use Case Diagram

b. User Management

User management allows users to control all aspects of their accounts and will provide administrators with tools to manage multiple user accounts. Users will have the ability to login and modify or delete their accounts.

1.1 The *Create User Account* functionality will allow a user to create an account.

1.2 The *Modify User Account* functionality will allow a user to edit attributes associated with an account.

1.3 The *Delete User Account* functionality will allow a user to delete an account.

1.4 The *User Login* functionality will authenticate users with their usernames and passwords, enabling appropriate permissions.

1.4.1 Passwords will be a minimum of six and a maximum of twelve alpha-numeric characters in length.

1.4.2 Users will have the ability to change passwords.

1.4.3 Users will have the ability to contact the administrator when they have forgotten their passwords.

1.4.4 The system administrator will have the ability to delete user access and modify access levels.

1.4.5 Organizational managers will have the ability to modify or delete their organization's respective users' access.

c. Organization Management

Organization management will provide certain users with the means to control various aspects of their respective system accounts.

2.1 The *Manage Organization* functionality will allow a user to add, modify, or delete an organization's attributes, which includes its address, phone number, building number, or associated resources.

2.2 The *Manage Physical Location* functionality will allow a user to add, modify, or delete an organization's building, which is identified by its attributes, including stores, distribution centers, and headquarters.

d. Resource Information

Resource Information allows users to submit information to the system that describes the status of resources and resource needs.

3.1 The *Status of Resource* functionality allows users to describe resources in transit, delivered resources, or resources that are needed at a specific location.

3.2 The *Set Resource Utilization Defaults* functionality allows a user to set default numbers for utilization per person calculations, such as the amount of water needed per person, per day in a given environment.

e. Search

The *Search* will allow users to search information submitted by other users and organization query and view resource data and locations.

4.1 The *Query and Filter Data* functionality will allow a user to query data and return a data set.

4.2 The *View Map* functionality will allow a user to view the results of a query geographically on a virtual map. This function will include the ability to pan, zoom, or rotate the map to focus on any portion in greater detail.

4.3 The *Download* functionality will allow a user to download and name a set of search results as a KML file, which can be used with Google Earth or similar geospatial programs.

C. USE CASES

A compilation of all use cases and use case diagrams for SDR-1 can be found in Appendix A. An example use case is present in Figure 2.

1. Generate Search

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder.

Stakeholders and Interests:

- Users: During crises, users need a means of locating resources to build the Search in support of planning efforts.

- Individuals/private organizations: Requires the need to locate resources to build a Search to increase their situational awareness.

Preconditions:

- User is at the Search generation interface.

Postconditions:

- A Search has been generated.
- All locations and their associated resources are depicted on the Search.

Main Success Scenario:

1. User defines query criteria for and name for Search and submits it.
2. System stores the criteria and adds to a lists of save Searches.
3. User selects a Search from the list to view.
4. System returns a map showing the Searches.
5. System returns a test list of resources below the map.

Alternative Flows:

5b. User wants to adjust search and returns to Step 1.

Figure 2. Use Case Example

D. NON-FUNCTIONAL REQUIREMENTS

1.0 *Usability*

1.0.1 System shall comply with ISO usability standards.

1.1 *Learn ability*

1.1.1 Novice users should be able to create or modify an account within 120 seconds.

1.1.2 Users should be able to submit a need request within 120 seconds.

1.1.3 Users should be able to locate a resource on a virtual map within 60 seconds.

1.1.4 Novice users should be comfortable navigating the site within 10 minutes.

1.2 *Aesthetics*

1.2.1 The software shall make effective use of repeating visual themes.

1.2.2 The software shall be visually consistent even without graphics.

1.2.3 The site shall use moderate, consistent colors with easily readable fonts.

1.3 *Consistency*

1.3.1 Appearance and performance shall be consistent across all browsers.

1.3.2 The system will leverage object-oriented frameworks when possible.

1.4 *Documentation*

1.4.1 All documentation shall be available electronically within the software.

1.5 *Compatibility*

1.5.1 Software shall be compatible with modern Internet browsers including Firefox 3.0, Safari 4.0 and Explorer 7.0.

1.5.2 Software shall be compatible with HTML 5.0 Web standards.

1.5.3 Software will not require a third party plug-in such as Adobe Flash, Microsoft Silverlight or Google Gears.

1.5.4 Software shall work on all Microsoft XP, Vista, Windows 7.0, MAC OS 10.4 or later editions.

2.0 *Reliability*

2.0.1 The system shall be available 99 % of the time.

2.1 *MTTR*

2.1.1 Mean time to repair shall be less than 60 minutes.

2.2 *MTBF*

2.2.1 Mean time between failures shall be greater than 120 days.

2.3 *Accuracy*

2.3.1 The geospatial latitude-longitude shall be a float point with precision to six decimal places.

3.0 *Performance*

3.1 *Throughput*

3.1.1 Systems shall accommodate a minimum of 500 users simultaneously.

3.2 *Resource Consumption*

3.3.1 Computing resources should scale to accommodate high use situations.

3.4 *Response Time*

3.4.1 System shall have an average transaction time of less than 3 seconds.

3.4.2 Resource update shall be reflected on the map within an average of 15 seconds regardless of the bandwidth.

3.5 *Scalability*

3.5.1 The system shall have the ability to scale from a minimum of 500 users to a maximum of 15,000 simultaneous users, bandwidth dependent.

4.0 *Supportability*

4.0.1 All changes to software shall automatically be reflected across the entire software spectrum.

4.1 *Maintainability*

4.1.1 Corrective maintenance shall be performed in accordance with ISO standards.

4.2 *Installability*

4.2.1 The system shall require no additional installation beyond Web browser.

IV. DESIGN

A. INTRODUCTION

The requirements captured in the previous chapter were used as a roadmap to guide decisions, development, and high-level design of the proposed system. This chapter discusses concepts behind the design and describes the system in terms of its model-view-controller architecture.

B. SYSTEM CONCEPT

This is not an inventory or supply-chain management system. Based on the LOGCOP concept, its purpose is to improve disaster response by providing an integrated and interactive "neutral zone" where individuals and organizations can share information openly and transparently across all levels of the private and public sectors. It accomplishes this through a Web-based interface that organizations can use to voluntarily describe the disposition of resources without requiring direct communication between the organizations themselves.

Through SDR-1, public and private organizations can abstractly describe resources within their stores, distribution centers, or warehouses without disclosing detailed information that might jeopardize security or competitive advantage. Instead of providing inventories, which include brand, SKU, quantity, etc., organizations will use Web-based forms to describe resources in abstract terms, such as five pallets of bottled water or three 200-watt power generators. This approach is less of a deterrent to companies that want to participate but cannot afford to share detailed information. Once information is submitted through a lightweight, Web-based interface, it is stored in a data repository. The stored information describes resources at different locations and is available to all participating organizations through the same Web-based interface and can be used as a starting point for a rapid, coordinated response.

SDR-1 also provides a means to coordinate resource distribution across organizations using self-synchronization. Responding organizations share information by using simple, Web-based forms to describe the status of resources at specific locations. For example: resources delivered to Monterey High School or needed at Dodger Stadium. Each organization can adjust their individual distribution based on shared information in the system.

Organizations can use the Web-based system to define search criteria such as location, time, or resources. Results can be returned as text, points on a virtual map, or a KML file, which can be imported into another system. Since SDR-1 does not track inventories for any organization, a delivery of resources will not cancel out or correspond to a similar request for resources from the same location. They are two discrete pieces of information that help describe the situation at a given location. The two pieces of information would show up side-by-side if placed on a virtual map. Calculations to derive net values are planned for future versions of the system.

C. CLOUD COMPUTING FOR SYNCHRONIZING DISASTER RESPONSE

Cloud computing is well-suited to the proposed application and the disaster response domain. Cloud based applications do not actually run on any of the available servers until they are needed. When a user types a Web address into a Web browser, an instance of the application is started on one of the thousands of available servers and continues to run in memory as long as users are requesting it. As more users request the site, additional instances are started on different servers to handle the increased traffic. When traffic slows the various instances are terminated, freeing the computing resources for other applications. There are three key benefits to this approach:

1. Data and computing resources are distributed geographically, providing the ability to run an application from any available data center in any region. This is important since the location of a future disaster is impossible to predict. The application will continue to run even if data centers in the region are destroyed.

2. Applications scale automatically to accommodate hundreds of thousands of concurrent users. This is another important characteristic for a disaster response system.

3. Compute resources are charged only as they are used. Development, testing, and normal usage require very few computing or data resources. Greater costs will not occur unless the application is heavily used in response to a disaster at which point it will be justified. This is more reliable than purchasing a small set of servers that may be overwhelmed during a disaster and less expensive than purchasing and maintaining dedicated computer clusters that would remain idle the majority of the time.

D. APP ENGINE PLATFORM OVERVIEW

Our proposed application is designed to take advantage of the Google App Engine platform, which provides robust infrastructure comprised of thousands of servers and petabytes of data storage. Interaction between the App Engine components is illustrated in Figure 3, which follows an HTTP request through the system.

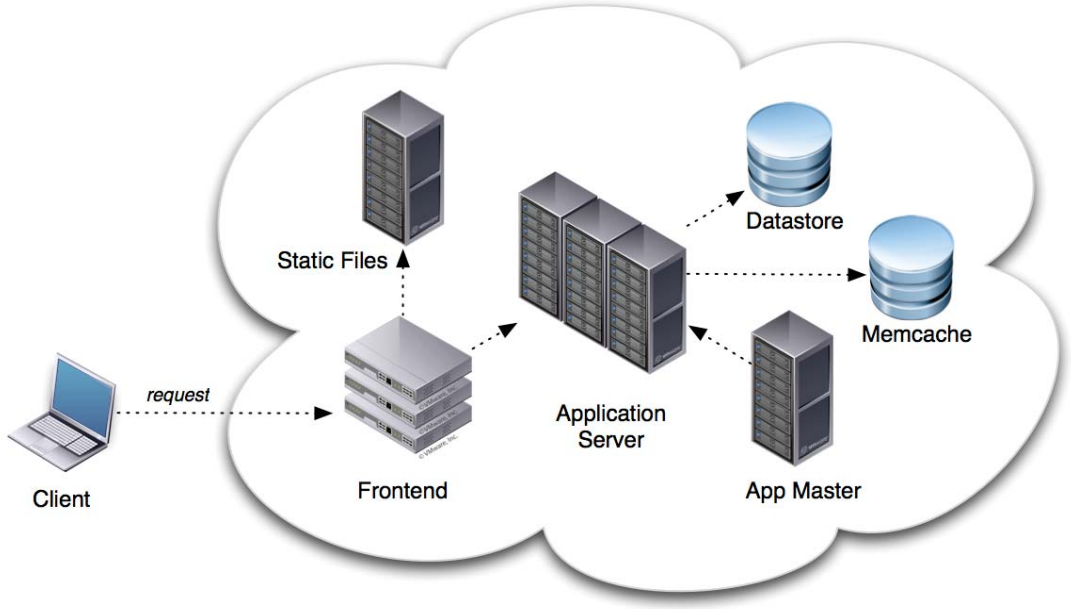


Figure 3. Physical Architecture (After Sanderson, 2010)

Incoming requests are received by the frontend, which determines the intended application based on URL specified. Once the destination is identified, the frontend evaluates whether the request maps to one of the application's static files handled by a set of dedicated static file servers or if it requires processing through the application itself. Static files such as pictures are returned to the client directly by the static file servers. Requests that need processing require an instance of the application to be running on one of the available application servers. A new instance, based on the App Master, is started on an available application server if no instances are already running. More instances are started as required to handle additional traffic. The request is directed to the instance, which interacts with persistent data in the Datastore or the memory cache, performs necessary operations, prepares a response to the request and returns it to the client (Sanderson, 2010).

E. PROPOSED APPLICATION (SDR-1)

The proposed application separates development concerns into three primary areas: model, view, and controller. The model refers to persistent data kept in the Datastore. The view consists of the HTML templates, CSS, and Javascript that define the user interface, and the controller is comprised of event handlers within the Python code that orchestrate interactions between the model and view. This separation allows changes in one area without requiring changes to another, providing flexibility and updatability (Severance, 2009).

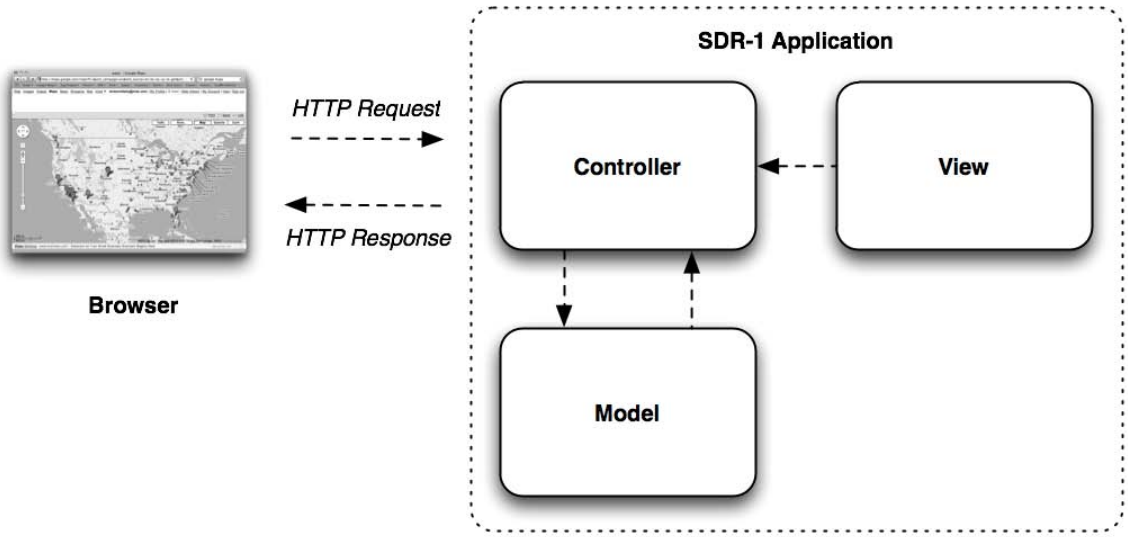


Figure 4. Model-View-Controller (After Severance, 2009)

The SDR-1 architecture is based on the Model-View-Controller pattern, with each element deployed to different physical servers in the cloud-computing environment. The architecture consists primarily of a Web browser running a user's personal computer and the SDR-1 application which consists of a model, view, and controller. The controller and dynamic aspects of the view are deployed together and instantiated on available application servers to handle incoming requests, while static portions of the view are deployed on servers optimized to handle static files. SDR-1's model resides in the Google Datastore, which sits atop Google's Bigtable, a distributed data repository. As depicted in Figure 5, the controller depends on the model and view to provide data and structure, respectively, before it can respond to the incoming request.

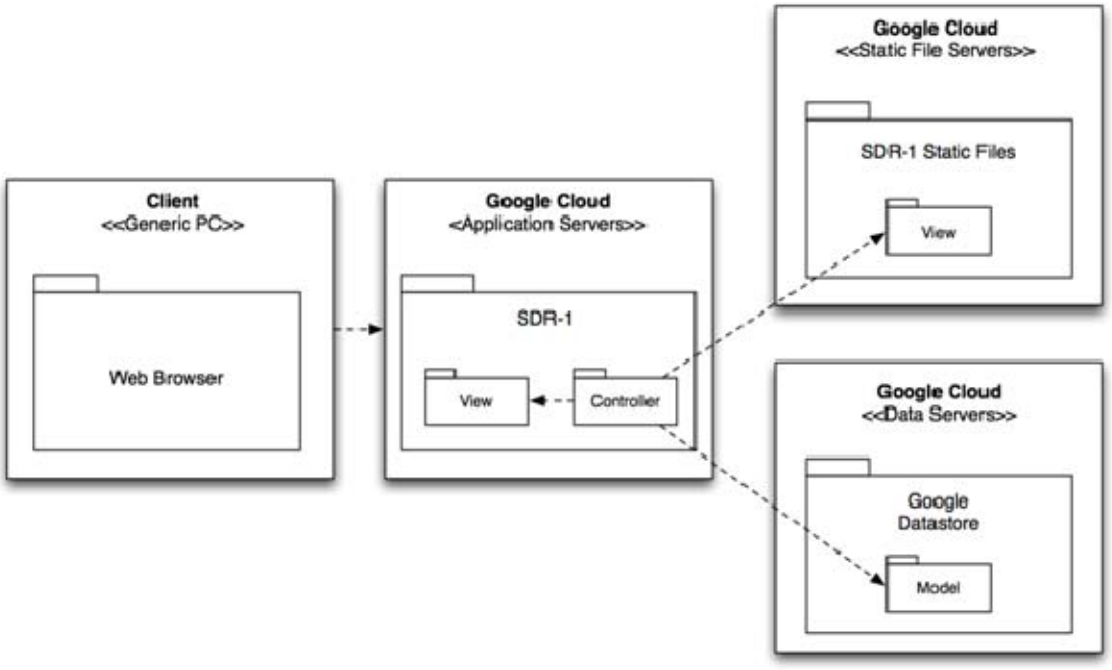


Figure 5. SDR-1 Deployment

1. Model

This section describes the structure of data that persists in the Datastore over time.

a. Datastore

The Datastore is neither a relational nor object database but a complex distributed data storage system based on Google Bigtable. The schemaless object Datastore has a query engine, atomic transaction support, and a robust data modeling API. It is set apart from traditional databases by an architecture that enables automatic scaling to petabytes of data. Datastore makes extensive use of indexes for high performance reads across thousands of physically distributed machines. This replaces the relational approach of normalizing data and joining tables to store information with a distributed approach, which stores information in rows, not tables. Additional information on the Datastore can be found in Appendix C.

b. Proposed Classes

Using requirements from the previous chapter, data models were developed to capture real-world objects needed for the proposed disaster response system. The identified objects fall into one of seven classes: User, Profile, Organization, Location, Resource, Disaster Area, and Search.

1. User Class

The User class, Figure 6, is controlled by Google and represents users across the Google ecosystem. When a user signs up for Google Apps their information is stored in their Google account, which only Google and the user can access. Google does provide a service that allows App Engine applications to determine if a user has been successfully authenticated using their Google Account or OpenID. Once signed-in, the service provides SDR-1 access to selected user properties including: nickname, e-mail, and user ID, the federated user ID and federated provider for OpenID users. We leverage these services to take advantage of the existing Google infrastructure as well as the OpenID platform now being piloted by the Government Services Administration (Thibeau, 2009). A detailed description of the User service provided by Google and OpenID can be found in Appendix D.

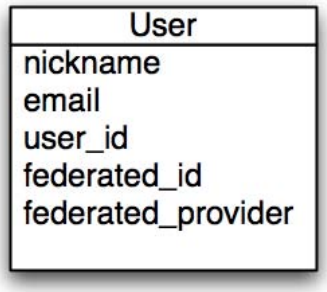


Figure 6. User Class and Properties

2. Profile Class

Since our application cannot directly access Google's User class, another class is needed to store additional user information need by SDR-1. The Profile class, Figure 7, fulfills this function by capturing properties including: first name, last name, phone number, organization, and authorization level. The Profile also stores the

user ID of the associated user. A password is not listed among the included properties since it is handled through the authentication services provided by Google. The proposed system is designed to have four levels of authorization, levels 0 through 3, which define a user's ability to input or edit data associated with a specific organization, such as defining store locations or resources within those stores.

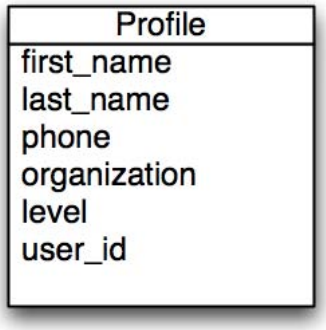


Figure 7. Profile Class and Properties

3. Organization Class

All public and private organizations are represented by the Organization class, Figure 8, which includes the properties: name, URL, category, and user ID. The name and URL refer to the official organization name and Web site respectively. They are used to ensure uniqueness of the organization within the system. The category describes the organization as: government, commercial, non-profit, faith-based, or volunteer. The user ID associates the organization with the user who last submitted the information describing the organization. This provides a means to trace changes to a user within the organization. Other information includes: phone numbers, physical addresses, and e-mail addresses. This information is associated with physical locations and is handled through the location class.

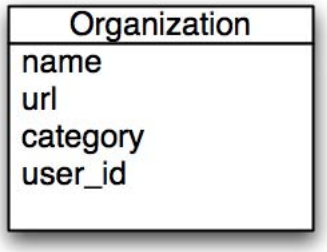


Figure 8. Organization Class and Properties

4. Location Class

A location captures information describing a geographic position and its associated resources. A location might be a distribution center, warehouse, organization headquarters, or retail store. A location could also refer to a position in the disaster area where resources are required, resources are delivered, or resources are en route. The goal of the location class, Figure 9, is to collect pertinent information in the most streamlined manner, limiting input required by users. Properties needed to describe a location include: address, FEMA region, lat-long, location type, resources, organization, and user ID. The address, FEMA region, and lat-long describe the physical position. The type is selected from headquarters, warehouse, distribution center, store, request, delivery, or transit. The user ID and organization are associated with a user who submits the location. The resources property contains a reference to each of the resources associated with this location, defining a one-to-many relationship between a location and resource. As stated previously, the Datastore is not a relational database. The resources property is actually a list reference property, each one providing a link to one resource associated with the location. The user's authorization level determines what information a user can submit. All users can submit request, delivery, and transit locations while only level-2 and level-3 users can add headquarters, warehouses, distribution centers or stores. It is important to note that all resources associated with a request location type are treated as negative numbers, while all other location types are treated as positive numbers. Transit location types refer to the ultimate destination where resources are intended they do not capture the transit route. They are differentiated from delivery location types by color in the view.

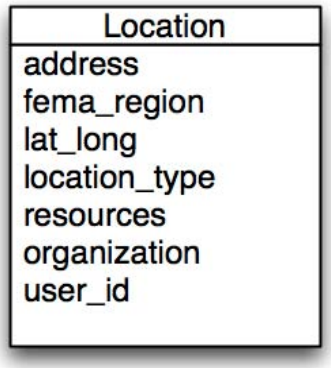


Figure 9. Location Class and Properties

5. Resource Class

The Resource class, Figure 10, is designed to capture information describing resources in abstract terms, avoiding unnecessary details such as brand or stock-keeping unit (SKU). Each resource is associated with a specific location and described using seven properties: resource type, palletized, number, weight each, unit of measure, quantity each, and quantity total. The resource type is currently limited to one of eight choices representing FEMA's Big-8 resources. Future iterations could expand the resource choices but not at the expense of usability. The palletized property is a Boolean resolved to true if the resource is on standard pallets. Number refers to the number of standard pallets for palletized resources or the number of individual items if they are not palletized. Weight is also measured per pallet or per individual item for items not palletized. The unit of measure captures the unit that best describes the item. For water, the unit of measure would be liters. For power generators, it would be watts. For blankets, it would be individual units. The quantity each describes the amount per pallet or per item in terms of the unit of measure. The quantity total is the result of the quantity each multiplied by number of pallets or number of individual units, depending on whether it is palletized.

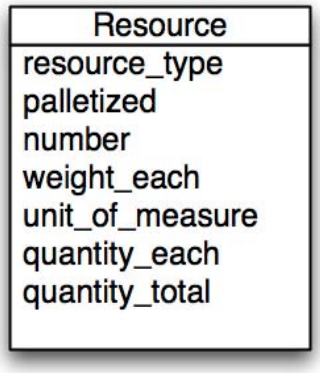


Figure 10. Resource Class and Properties

6. Search Class

The search class, Figure 11, is used to define a set of search criteria used as filters when querying the Datastore for locations and their associated resources. Searches can also be shared with all members of an organization. It is important to understand that search results are not being stored or shared. Only the criteria used to define the search is stored. This approach is designed to eliminate repetition by allowing a user to define the criteria for a search and use it many times as the situation evolves. The ability to share a search is designed to prevent redundant work within an organization. Search properties include a user defined name, a Boolean property that resolves to true if the search is shared across the organization, as well as the user ID and organization of the user who defined the search. Additional properties that define filter criteria include time, organization, resource type, location type, FEMA region, and address. Incomplete address such as state or zip can be submitted.

Figure 11. Search Class and Properties

7. Disaster Area Class

The DisasterArea class, Figure 12, is used to describe a geographical area designated by FEMA or USNORTHCOM following a disaster. Properties include a name for the designated disaster area, a list of points that define it, a Boolean to determine if it is shared to all other users, and the user ID of the individual who submitted it.

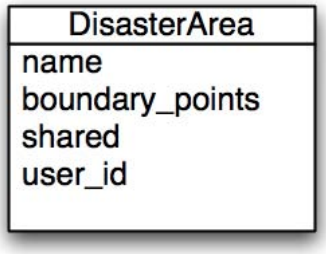


Figure 12. DisasterArea Class and Properties

The diagram in Figure 13 shows the relationships between the various classes using standard UML notation. For example: At least one but possibly many users belong to one organization.

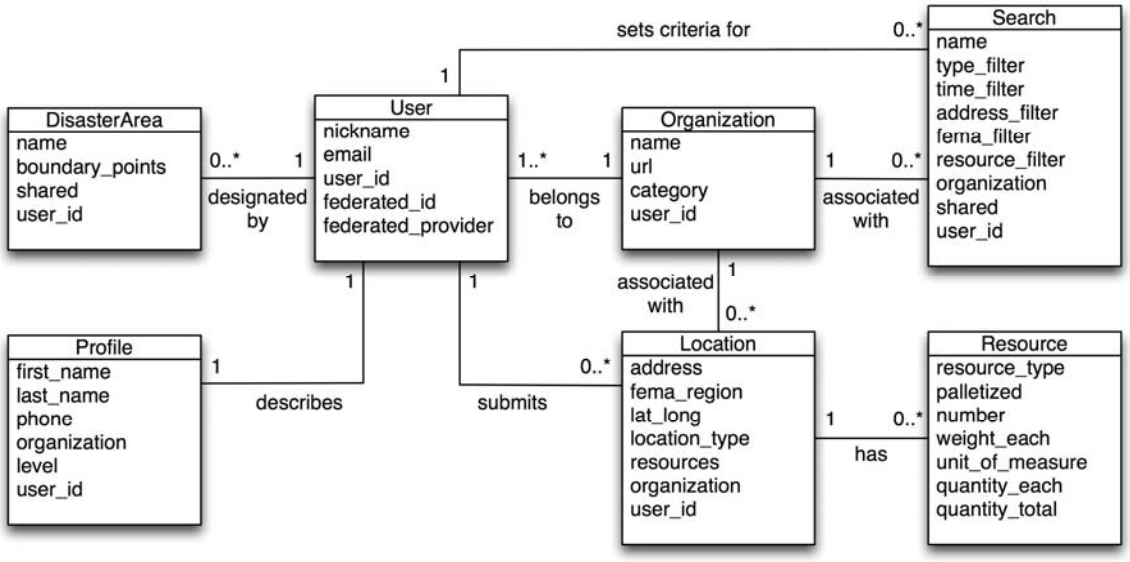


Figure 13. Entity Relationships

8. Model Class

Individual classes must persist in the Google Datastore. This is accomplished through the Model class provided through Google App Engine, which already contains methods to take data obtained from a user and put it in the Datastore. Each of the SDR-1 classes is a subclass of the Model class and can perform all operations of the Model class. This includes operations to get data objects from the Datastore and put data objects into the Datastore or perform queries over all objects in the class. The diagram in Figure 14 captures all of the operations that belong to the Model class and extend to the SDR-1 sub-classes. Detailed information on the Model class and Datastore API can be found in Appendix C. The Model class is depicted in Figure 14.

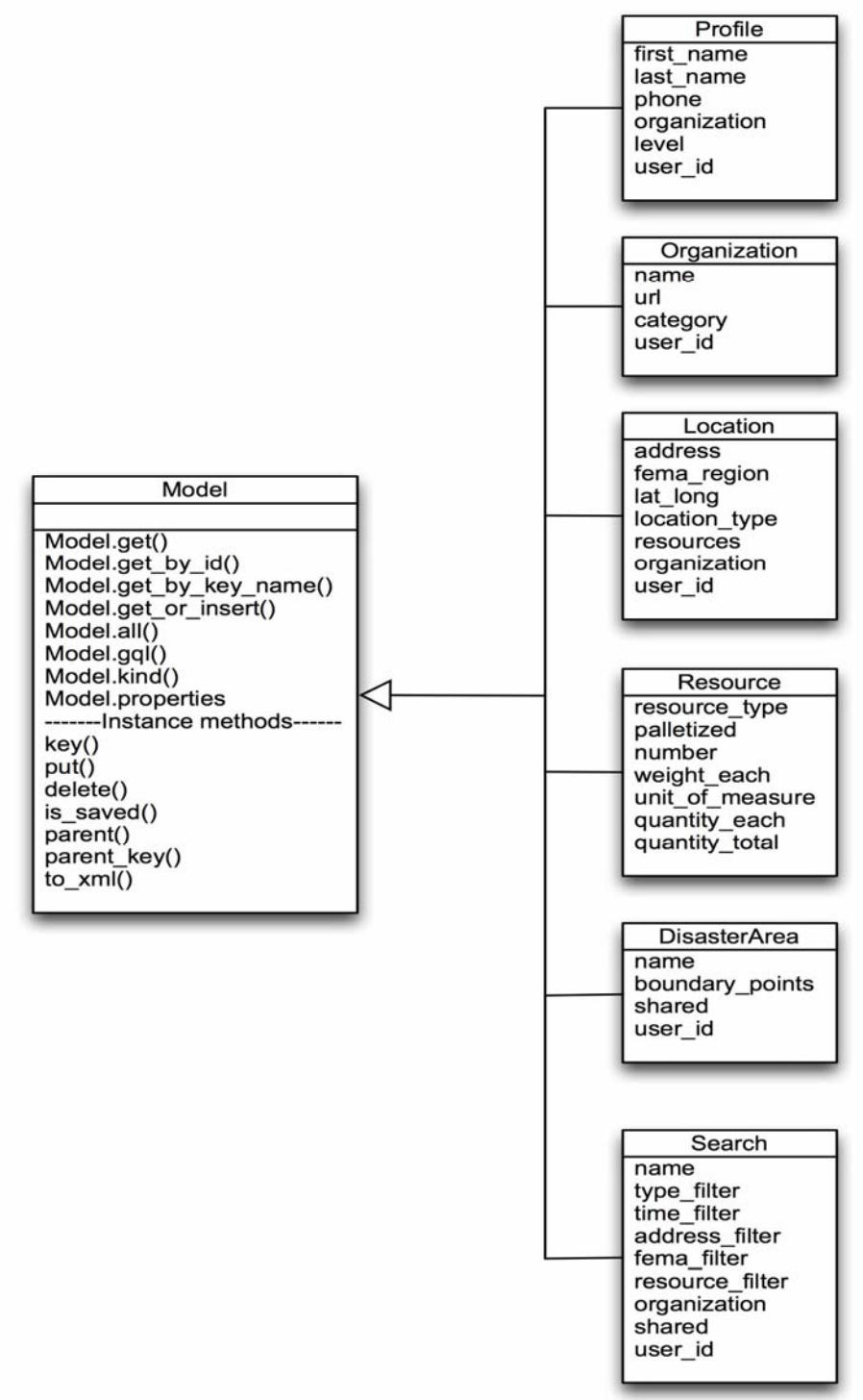


Figure 14. Model Class and Subclasses

2. View

The view consists of a set of HTML templates and cascading style sheets that define the look and feel of the proposed application. When the controller needs a view generated, it calls on the template engine to create it. A template engine is included as part of the Web application framework of the App Engine Platform.

a. Template Engine

The template engine, Figure 15, merges data retrieved from the model with appropriate HTML templates to generate a view that can be returned to the user. The path to the template and values to be inserted into the template are passed as arguments from a request handler in the controller to the template engine, which replaces tagged variables in the template with data retrieved from the model to produce the Web page, as illustrated in Figure 15 (Sanderson, 2010).

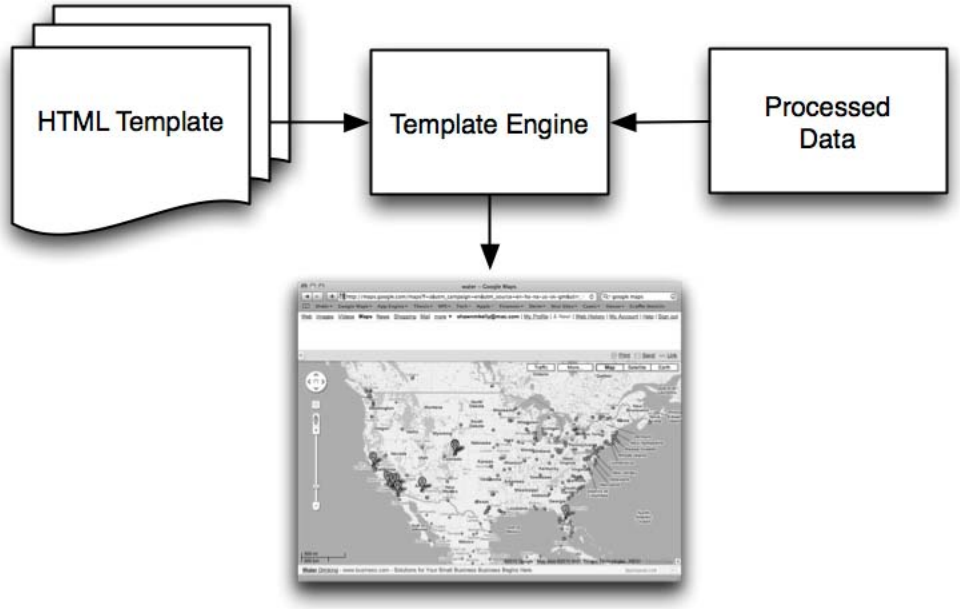


Figure 15. Template Engine (After Sanderson, 2010)

b. HTML Templates

The structure of the view depends on a set of HTML templates, Figure 16, which describe content presented to the user through the hypertext markup language. Inheritance and aggregation are supported by the template engine. The templates allow the content to be divided into a hierarchy of extendable and reusable templates, which are combined to create a coherent Web page for the user. The header, footer, and navigation elements are common to every Web page within the application and represented through a single, reused "Base" template. Changes made to this template are reflected across all pages that extend them. A series of individual page templates extend the base template, providing additional content to describe each page. Detailed illustrations and explanations of the templates can be found in Appendix B.

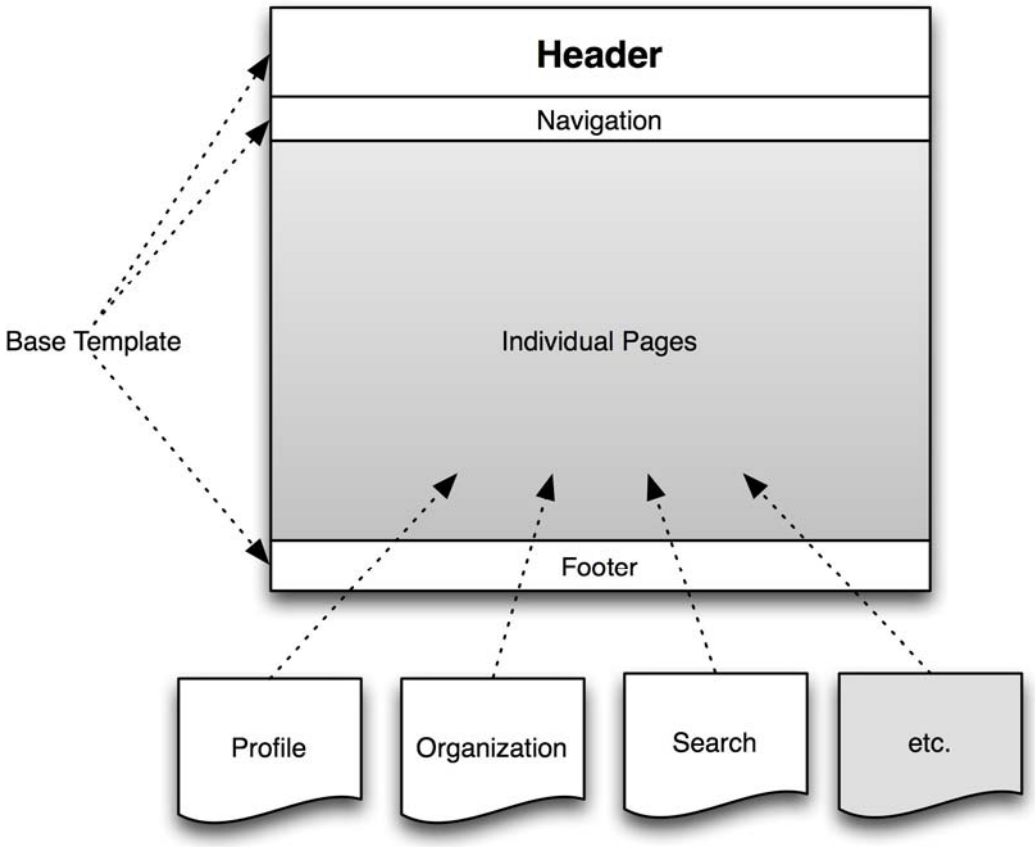


Figure 16. HTML Template Layout (After Severance, 2009)

c. Cascading Style Sheets

Cascading style sheets (CSS) are used to separate the appearance of the application Web page with regard to colors, fonts, and layouts from the structure written in HTML. Imbedding the style within each HTML page would reduce maintainability since style changes would need to be repeated on every page to achieve consistency. CSS resolves the problem by providing style definitions that are shared across multiple pages. This approach eliminates redundant code, resulting in consistent Web pages that require less maintenance (Bert, Celik, Hickson & Lie, 2009). The presentation for SDR-1 is described through a single cascading style sheet shared by all HTML templates. The style is sent from a static file server directly to the client to provide the correct presentation details to the browser as it interprets the HTML returned by the system.

3. Controller

The controller contains the logic of the SDR-1 application. It receives client requests through the frontend and routes them to the appropriate handler, which performs operations and interacts with the model and view before responding to the request. In the case of SDR-1, the controller consists of a configuration file in addition to a series of request handlers.

a. Configuration File

In order for the frontend to direct incoming traffic, it depends on the configuration file to provide a route to the appropriate request handler based on URL paths included in a request. This mapping is specified in a YAML file that tells the frontend which python script needs to be loaded onto the application server in order to handle the request. This file also defines how long the application should be cached in memory on the application server and whether the HTTP connection needs to be secured (Sanderson, 2010).

b. Request Handlers

A series of request handlers are responsible for dealing with incoming HTTP requests for different URLs within the application. In some cases, more than one URL is mapped to a single handler which analyzed the URL to see how to respond to it. Request handlers contain the code for exchanging data with the model and view in order to respond to the user. When an incoming request is received by the application, an instance of the request handler associated with the URL is created. The instance performs the necessary operations such as getting information from the Datastore and rendering a page before it returns the response and terminates (Google App Engine, n.d.).

Google provides a base class called RequestHandler to deal with incoming requests. However, it does not achieve everything needed by the SDR-1 application. To resolve this, a new subclass called MainHandler was created. MainHandler extends the RequestHandler class by adding methods to assess user privileges and call for pages to be rendered. These methods are available to the SDR-1 subclasses, which extend MainHandler to IndexHandler, ProfileHandler, OrgHandler, OrgMembersHandler, LocationHandler, SearchCriteriaHandler and SearchResultsHandler. This hierarchical approach allows the functionality to be defined at the appropriate level, reducing repetition and likelihood of mistakes. The organization of these classes is illustrated in Figure 16 and further described in Appendix C.

1. RequestHandler Class

The RequestHandler class, provided by the Google App Engine platform as part of the Web application framework, serves as the base class for all of SDR-1's request handlers. It has two attributes: one to hold the request and one to hold the response. It also has several methods to handle a variety of HTTP requests. A complete list of methods provided by the RequestHandler class is shown in Figure 17 (AppEngine API, 2010). Detailed information describing the RequestHandler class is included in Appendix C.

2. MainHandler Class

The MainHandler class extends the RequestHandler class by adding four new methods. The first two, isNewUser and isValidUser, are used to add new users or authenticate existing users, respectively. App Engine provides built in support for designating administrators. The isAdminUser method checks to see if the current user has administrator permissions. The final method, called renderPage, substitutes the appropriate values into a designated template so the rendered page can be returned to the user.

3. SearchResultsHandler Class

This class extends the MainHandler class and provides a significant amount of SDR-1 functionality beyond querying the Datastore and returning filtered, sorted results. It contains the necessary functions to pass latitude-longitude information needed by the view to produce a map using MapRender.js. Additionally, it contains a function to convert query results to KML, which can be sent to the e-mail address of the currently authenticated user.

4. LocationHandler Class

The LocationHandler class extends the MainHandler class and also has PUT and GET methods adapted to work with incoming requests and data dealing with Locations. The handler must call the geocoding Web service to translate addresses to lat-long coordinates or from coordinates to an address if one exists. If a given address cannot be resolved to a set of coordinates, the user must be asked to provide a different address. A confirmed address is required for every location with a structure, but latitude-longitude coordinates are sufficient for request, delivery, and transit locations. Geocoding is provided through Google Services and documented in the Google Maps API in Appendix D.

5. Remaining classes

The remaining classes all extend the MainHandler class, using the same techniques for authenticating users and rendering pages. Each subclass has private attributes and its own PUT and GET methods adapted to the respective requests. These

classes have no special functionality beyond providing a means to interact with the Datastore and view in order to return the appropriate response to the user.

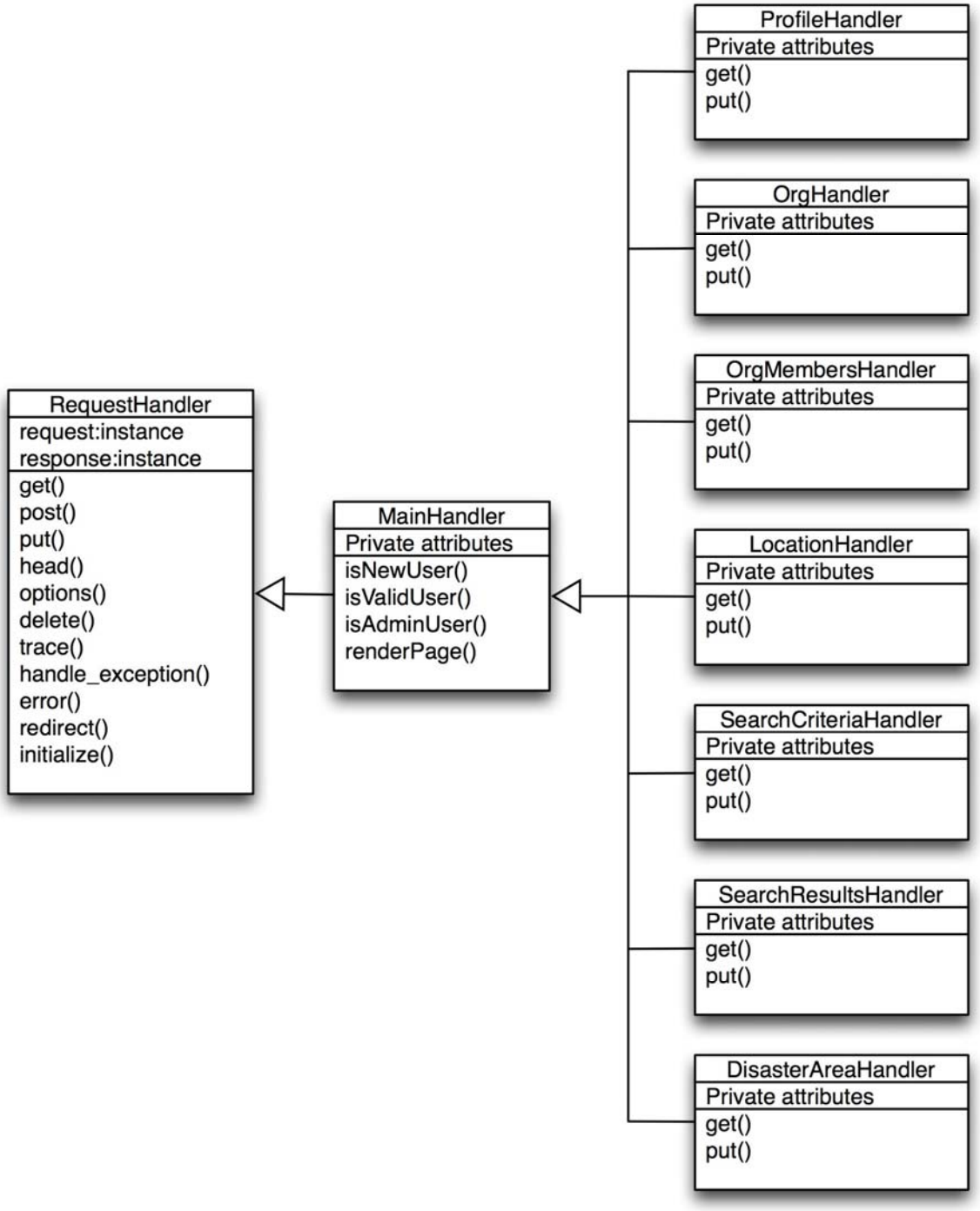


Figure 17. RequestHandler Class and Subclasses

F. FEEDBACK

A member of the USNORTHCOM working group in charge of developing software solutions to enhance disaster response collaboration evaluated the SDR-1 design and provided several pages of detailed feedback, which highlighted key benefits of the proposed system and identified required areas for additional research. The complete memorandum can be found in Appendix E.

The memorandum stated that the cloud computing concept implemented on the App Engine Platform is an excellent solution for the proposed system since it will potentially keeps costs low and ownership neutral, which is a key element of the system. The approach to handling locations was also appreciated, since it addressed request and distribution information in addition to resources located within physical structures such as stores. From a logistics point of view, the use of pallets as a standard way to measure resources was notes as, "a thoughtful and necessary inclusion for system implementation." The ability to store search criteria and the use of a simple HTML interfaces were also appreciated.

Several items that need to be addressed in future iterations were also discussed. The system will eventually need to expand beyond FEMA's Big 8 resources and provide some method for non-standard resources to be identified. More flexibility is required within the search, allowing wider ranges when filtering by time and providing more options for sorting results. The memorandum suggested that results could auto update every few minutes without interaction from the user. A detailed analysis of permissions was also requested.

Overall, the memorandum concluded, "This is a thoroughly researched and well written thesis that needs a prototype built on its foundation soonest so that the program can move forward after a hand-on test." The complete memorandum can be found in Appendix E.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

1. SUMMARY OF WORK

This thesis examined current disaster response systems in order to discern and leverage their benefits for attaining synchronization in disaster response domains. The reviewed systems and architectures, which work well in the commercial sectors, do not provide a means for effective collaboration between organizations required for disaster response. The thesis also reviewed business complexities that affect organizations' willingness to work together during disaster response. Stakeholders require a neutral zone in which information can be voluntarily and openly shared between responding entities, both public and private, without compromising security or competitive advantage. An in depth review of stakeholders and their respective needs was conducted and translated into a prioritized set of features, which were the foundation of hardware and software requirements. A high level design was developed based on the research and stakeholder requirements. The proposed design was submitted to USNORTHCOM, the primary stakeholder for feedback. USNORTHCOM highlighted system strengths, identified concerns, and provided a path forward.

2. KEY FINDINGS AND CONTRIBUTIONS

A significant finding was that many communication breakdowns in the disaster domain occur due to a lack of interoperability between the organizations' information systems. Additionally, current systems often become ineffective as client-server architectures develop bottlenecks during peak usage. While Web services and service-oriented architectures provide potential solutions to the interoperability problems, they do not address surging traffic and degraded infrastructure common to a large disaster. This scalability problem is a big issue because large disasters require systems that can handle significant volumes of data even when infrastructure has been partially destroyed. Research revealed that cloud computing, which can take advantage of Web services and service-oriented architectures, is resistant to catastrophic failure due to its distributed

nature. The cloud computing architecture is built around the concept of elasticity, allowing cloud-based applications to scale across thousands of servers to handle incoming traffic. These abilities are critical in the disaster response domain.

The main contribution of this thesis is a high-level design for a proposed system named Synchronized Disaster Response 1, for providing a means to synchronize disaster response through shared awareness of the initial and evolving logistical situation in the disaster area. The SDR-1 design is predicated on the USNORTHCOM LOGCOP's conceptual model and the aforementioned critical requirements, which resulted from extensive background and research. The design prototype fosters a new model for accomplishing disaster response. It is developed from the mindset of minimizing barriers to voluntary participation by capturing resource information in terms that do not threaten security or competitive advantage. It is also one of the few current system designs created solely for enhancing disaster response communication and synchronization. This system is designed to reside on Google's cloud and take advantage of elasticity and redundancy provided by thousands of servers in Google datacenters around the world.

The system accomplishes its objective by providing organizations with a way to share information describing resource stockpiles during the mitigation phase, providing responders insight to potential resource availability prior to a disaster. During disaster response SDR-1 fosters information sharing through a lightweight user interface that helps organizations submit information describing resource needs and the movement of resources in the disaster area. Any individual or organization can use a standard Web browser to access SDR-1's search function and build a logistical picture describing the resource situation in a particular area. Search criteria can be defined once and used many times to provide the most current information. By consulting shared data that describes evolving logistical situations, organizations can determine where to deploy their assets for maximum impact. By broadcasting their movements, organizations can build the situational awareness of other responding entities. Direct coordination between responding organizations is not required for a rapid and evenly distributed response.

It must be noted that all systems are simply mechanisms for accomplishing goals. SDR-1 will only be effective if responders at every level incorporate its use in their disaster response plans.

3. EVALUATION AND FEEDBACK

The proposed design was submitted to members of USNORTHCOM involved with their Logistical Common Operating Picture and familiar with disaster response for a comprehensive evaluation. Since these stakeholders originally outlined a conceptual model that SDR-1 was predicated upon, their insight and detailed feedback on the proposed design is important in determining the way forward. The feedback highlighted benefits of leveraging cloud computing for the proposed system and agreed with many design decisions including implementation of the location class, the use of pallets for a standard of measure, and the ability to store search criteria for future use. It also noted areas of focus for future work, including more options for setting search criteria and sorting results as well as a full review of permissions implementation. A memorandum containing the detailed feedback is included in Appendix E. It recommends the project move forward to a functional prototype.

4. RECOMMENDATION FOR FUTURE WORK

As noted in the evaluation feedback, the high-level design provides a sufficient foundation to begin iterative development. It recommends that future work be focused on developing a functional SDR-1 prototype that can be tested by USNORTHCOM, FEMA, and entities interested in disaster response. The initial prototype should incorporate concerns highlighted by the feedback in Appendix E, while holding all additional features for follow-on iterations, each with thorough testing by stakeholders to ensure the application is easy to use and the barriers to entry remain low.

After the system has been deployed and thoroughly tested, researchers can leverage Web services to extend the system to be used on mobile devices such as cellular phones and tablets to be used in conjunction with cellular or hastily formed networks in the disaster area.

Disasters will continue to occur. Computer technology is available to address many of the technical challenges and, if implemented properly, it can bring organizations together to solve common problems. The proposed SDR-1 design is a first step towards improving the disaster response community through synchronizing efforts. Future research, development, and participation are required to continue forward in the disaster response domain.

APPENDIX A. SYSTEM USE CASES

1. Generate Search

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder.

Stakeholders and Interests:

- Users: During crises, users need a means of locating resources to build the Search in support of planning efforts.
- Individuals/private organizations: Requires the need to locate resources to build a Search to increase their situational awareness.

Preconditions:

- User is at the Search generation interface.

Postconditions:

- A Search has been generated.
- All locations and their associated resources are depicted on the Search.

Main Success Scenario:

6. User defines query criteria for and name for Search and submits it.
7. System stores the criteria and adds to a lists of save Searches.
8. User selects a Search from the list to view.
9. System returns a map showing the Searches.
10. System returns a test list of resources below the map.

Alternative Flows:

5b. User wants to adjust search and returns to Step 1.

2. Add Organization

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

Public/Private Entities have an interest in contributing information to be used in generating Searches for emergency response. Owners, managers, and workers need a simply way to add organization information into the system.

Preconditions:

- User is logged into the system.

Postconditions:

- New organization is created in the system.

Main Success Scenario:

1. User submits organization name, address, type, and URL.

2. System validates organization as unique.

3. System updates and saves data describing the organization.

Alternative Flows:

2b. System is not able to resolve address to a Lat-Long. System notifies the user and returns them to Step 1.

2c. Organization already exists in the system. System notifies the user and returns to step 1.

3. Modify Organization

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

- Owners, managers, and workers need a simple way to modify organization information in the system.

Preconditions:

- User is logged into the system.
- User has permission to modify the organization.

Postconditions:

- Organization information is modified in the system.

Main Success Scenario:

1. User selects organization to be modified.

2. System displays organization attributes.

3. User overwrites organization attributes.

4. System prompts user for a password.

5. User submits password.

6. The system updates and saves data describing the organization.

Alternative Flows:

5b. Password fails to authenticate user. System notifies user and returns to Step 2.

4. Delete Organization

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

- Users must be able to remove organizations from the system when the organization no longer exists or wants to participate.

Preconditions:

- Organization must exist in the system.
- User must have permission to delete organization.

Postconditions:

- The organization is removed from the system.

Main Success Scenario:

1. User logs in.

2. User selects option to delete the organization.

3. System displays list of organizations.

4. User identifies the organization to be deleted.

5. System requests confirmation with a password.

6. User submits password.

7. Organization is deleted from the system.

Alternative Flows:

6b. Password fails to authenticate user. User is prompted to resubmit password or quit.

5. Add Store

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

- Public/Private Entities: A prime consideration is that owners and managers need the ability to create their organizations' distribution centers in the system.

Preconditions:

- Organization must exist in the system.
- User must have permission.

Postconditions:

- New distribution center is created in the system.
- The distribution center is associated with an organization.

Main Success Scenario:

1. User selects option to add new distribution center.

2. System displays information fields for new distribution center.

3. User submits the distribution center information.

4. System validates distribution center and resolves address to a Lat-Long.

5. System adds the distribution center to the system.

Alternative Flows:

4b. System is not able to resolve address to a Lat-Long. User is directed to correct the address. Returned to Step 3.

4c. Distribution center already exists in the system. User is directed to update the existing system.

6. Add Resource Location

Scope: SDR-1.

Level: User-Goal.

Primary Actor: User.

Stakeholders and Interests:

All actors require accurate location information in order to maintain an accurate search for forecasting, planning and distributing resources.

Postconditions:

- New resource request or location created in the system.

Main Success Scenario:

1. User logs in.

2. User selects option to add a resource location.

3. System displays information fields for resource request.

4. User submits location, POC, phone number, resources, and quantities.

5. System validates information.

6. System displays submitted information back to the user for confirmation.

7. System captures date, time, and IP address of user.

8. Resource request is added to the system.

Alternative Flows:

5b. Information is invalid. User is prompted to correct information.

7. Edit Resource Location

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder.

Stakeholders and Interests:

- All actors require accurate location information in order to maintain an accurate search for forecasting, planning and distributing resources.

Postconditions:

- Resource location edited in the system.

Main Success Scenario:

1. User logs in.

2. User selects option to edit resource location.

3. System queries the user for the location identification number.

4. User submits the location identification number to the system.

5. System displays information fields for the resource location.

6. User edits the location.

7. System validates information.

8. System requests confirmation from the user.

9. System captures the submitted data, date, time, and IP address of user and

10. System modifies the respective location.

Alternative Flows:

7b. Information is invalid. User is prompted to correct information.

8. Create User Account

Scope: SDR-1.

Level: User-Goal.

Primary Actor: User

Stakeholders and Interests:

- Distinct users accounts are needed to provide proper permissions, security, and communication within the system.

Postconditions:

- A new user is created in the system.
- User is associated with an organization.

Main Success Scenario:

1. User selects option to create a new user account.

2. System displays information fields for account request.

3. User submits first name, last name, username and password.

4. System ensures username is unique.

5. System captures user data, date, time, and IP address.

Alternative Flows:

4b. System notifies user that username already exist and then returns to Step

9. Modify User Account

Scope: SDR-1.

Level: User-Goal.

Primary Actor: User.

Stakeholders and Interests:

- Once a user account is created, users need to update information within the system so it accurately describes the user.

Postconditions:

- A user's information is modified in the system.

Main Success Scenario:

1. User logs in.

2. User selects option to edit user account.

3. System displays information account fields.

4. User edits the information and confirms with password.

5. System captures updated data, date, time, and IP address of user.

Alternative Flows:

5b. Password fails to authenticate user. User is prompted to resubmit password.

10. Delete User Account

Scope: SDR-1.

Level: User-Goal.

Primary Actor: User.

Stakeholders and Interests:

- All users who wish to no longer participate in the system need a way to delete its information.

Preconditions:

- User must exist in the system.

Postconditions:

- The applicable user is removed from the system.

Main Success Scenario:

1. User logs in.

2. User selects option to delete the user account.

3. System requests confirmation with a password.

4. User submits password.

5. User is deleted from the system.

Alternative Flows:

5b. Password fails to authenticate user. User is prompted to resubmit password.

11. User Login

Scope: SDR-1.

Level: User-Goal.

Primary Actor: User.

Stakeholders and Interests:

- Users who wish to access the system need to have a means to access the system with minimal personal intrusion and maximum simplicity and expediency.
- System owners need a mean of validating users and setting access levels in order to protect user and organization information from non-respective entities.

Preconditions:

- User account exists.

Postconditions:

- User is logged into the system.

Main Success Scenario:

1. User submits user identification and password.

2. System authenticates user and displays the main page.

Alternative Flows:

2b. System fails to authenticate and returns to step 1.

12. Modify Store

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

- All users require correct distribution center information in order to maintain an accurate search for forecasting, planning and distributing resources.

Preconditions:

- Organization must exist in the system.
- User must have permission.

Postconditions:

- Distribution center information is modified.

Main Success Scenario:

1. User selects distribution center and chooses option to modify distribution center.

2. System displays the distribution center information fields.

3. User edits the information and submits it to the system.

4. System validates information.

5. System requests confirmation with a password.

6. User submits password.

7. System captures the date, time, and IP address of user.

8. System updates the distribution center data.

Alternative Flows:

4b. System is unable to resolve address to Lat-Long. User is directed to correct the address.

7b. Password fails to authenticate user. User is prompted to resubmit password.

13. Delete Store

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

- Managers and administrators need the ability to delete distributions submits that no longer exist.

Preconditions:

- Organization must exist in the system.
- User must be a member of the organization.

Postconditions:

- Distribution Center data is removed from system.

Main Success Scenario:

1. User logs in.

2. User selects a distribution center and option to delete it.

3. System requests confirmation with a password.

4. User submits password.

5. System deletes distribution center.

Alternative Flows:

5b. Password fails to authenticate user. User is prompted to resubmit password.

14. Add Default Resources

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

- Users need the ability to create default resource quantities and amounts that will be reflected in each store within the organization.

Preconditions:

- Organization must exist in the system.
- User must have permission.

Postconditions:

- New resource is created in the system.

Main Success Scenario:

1. User selects option to add a resource.

2. System displays information fields to be filled in.

3. User submits information to create new resource.

4. Resource is added in the system.

5. System captures date, time, and IP address of user.

Alternative Flows:

4b. Information is invalid. User is prompted to correct information.

15. Modify Default Resources

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

- Users need the ability edit default resource quantities and amounts that will be reflected in each store within the organization.

Postconditions:

- Resource description is modified in the system.

Main Success Scenario:

1. User selects option to modify a resource.

2. System displays information fields to be changed.

3. User submits information to modify existing resource.

4. Resource is modified in the system.

5. System captures date, time, and IP address of user.

Alternative Flows:

4b. Information is invalid. User is prompted to correct information.

16. Delete Default Resources

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Manager, Administrator.

Stakeholders and Interests:

- Users need the ability to delete default resource quantities and amounts that will be reflected in each store within the organization.

Preconditions:

- Default resource must be in the system.

Postconditions:

- Resource description is modified in the system.

Main Success Scenario:

1. User selects option to delete a resource.

2. System displays resource to be deleted.

3. User submits information to delete existing resource.

4. System prompts user to confirm deletion with a password.

5. User submits password.

6. Resource is deleted in the system.

7. System captures date, time, and IP address of user.

Alternative Flows:

6b. Password fails to authenticate user. User is prompted to resubmit password.

17. Post Resources Needed

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to submit a quantifiable resource for a specific location and the approximate number of people the resource is expected to support.

Preconditions:

- Previously logged in.

Postconditions:

- Resource with the respective quantities, location and the approximate number of people the resource is expected to support is added to the system.

Main Success Scenario:

1. User selects option to add a resource.

2. System displays resource information fields to be filled in.

3. User submits information to create the resource.

4. System validates the resource.

5. Resource is created in the system.

6. System captures date, time, and IP address of user.

Alternative Flows:

4b. Information is invalid. User is prompted to correct information.

18. Modify Resources Needed

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to modify the resource list's resources, quantities, specific location, and/or the approximate number of people the resources is expected to support.

Preconditions:

- Previously logged in.

Postconditions:

- Resource list in the system contains the correct resources, quantities, specific location, and approximate number of people the resources is expected to support.

Main Success Scenario:

1. User selects option to modify a list of resources.

2. System displays resource list information fields to be modified.

3. User submits information to update the list.

4. System validates the list.

5. List is modified in the system.

6. System captures date, time, and IP address of user.

Alternative Flows:

4b. Information is invalid. User is prompted to correct information.

19. Delete Resources Needed

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to delete resource lists.

Preconditions:

- Previously logged in.
- Resource list is in the system.

Postconditions:

- Resource list is deleted from the system.

Main Success Scenario:

1. User selects option to delete a resource list.

2. System displays resource lists currently in the system.

3. User selects resource list to be deleted.

4. System prompts user to confirm deletion with a password.

5. User submits password.

6. System deletes resource list.

7. System captures date, time, and IP address of user.

Alternative Flows:

5b. Password fails to authenticate user. System prompts user to resubmit password.

20. Post Resources in Transit

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to submit a list of resources that are in transit to a specific location.

Preconditions:

- Previously logged in.

Postconditions:

- System updated with a list of in transit resources, quantities, and the estimated arrival time for a specific location.

Main Success Scenario:

1. User selects option to add in transit resource list.

2. System displays list of information fields to be filled in.

3. User submits information to create the list.

4. System validates the resource list.

5. In-transit resource list is created in the system.

6. System captures date, time, and IP address of user.

Alternative Flows:

4b. Information is invalid. User is prompted to correct information.

21. Modify Resources in Transit

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to modify a list of resources that are in transit to a specific location.

Preconditions:

- Previously logged in.

Postconditions:

- System updated with a list of in transit resources, quantities, and the estimated arrival time for a specific location.

Main Success Scenario:

1. User selects option to add in transit resource list.

2. System displays list of information fields to be filled in.

3. User submits information to create the list.

4. System validates the resource list.

5. In-transit resource list is created in the system.

6. System captures date, time, and IP address of user.

Alternative Flows:

4b. Information is invalid. User is prompted to correct information.

22. Delete Resources in Transit

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to delete a list of resources that are in transit to a specific location.

Preconditions:

- Previously logged in.

Postconditions:

- A list of in transit resources has been removed from the system.

Main Success Scenario:

1. User selects option to delete in transit resource list.

2. System displays list of in transit resource lists.

3. User selects in transit resource list to be deleted.

4. System prompts user to confirm deletion with password.

5. User submits password.

6. System deletes in transit resource list.

7. System captures date, time, and IP address of user.

Alternative Flows:

6b. Password fails to authenticate user. User is prompted to resubmit password.

23. Post Resources Delivered

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to submit a list of resources that has arrived at a specific location.

Preconditions:

- Previously logged in.

Postconditions:

- System updated with a list of resources and quantities that arrived at a specific location.

Main Success Scenario:

1. User selects option to add arrival resource list.

2. System displays list information fields to be filled in.

3. User submits information to create the arrival resource list.

4. System validates the arrival resource list.

5. Arrival resource list is created in the system.

6. System captures date, time, and IP address of user.

Alternative Flows:

4b. Information is invalid. User is prompted to correct information.

24. Modify Resources Delivered

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to modify a list of resources that has arrived at a specific location.

Preconditions:

- Previously logged in.

Postconditions:

- System updated with a list of resources and quantities that arrived at a specific location.

Main Success Scenario:

1. User selects option to modify arrival resource list.

2. System displays arrival resource list information fields to be filled in.

3. User submits information to update the arrival resource list.

4. System validates the arrival resource list.

5. Arrival resource list is modified in the system.

6. System captures date, time, and IP address of user.

Alternative Flows:

4b. Information is invalid. User is prompted to correct information.

25. Delete Resources Delivered

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Responder, Administrator.

Stakeholders and Interests:

- Responders need the ability to delete a list of resources that previously arrived at a specific location.

Preconditions:

- Previously logged in.

Postconditions:

- A list of resources that previously arrived was removed from the system.

Main Success Scenario:

1. User selects option to delete arrival resource list.

2. System displays arrival resource lists in system.

3. User submits information to delete the arrival resource list.

4. System prompts user to confirm deletion with password.

5. User submits password.

6. Arrival resource list is deleted from the system.

7. System captures date, time, and IP address of user.

Alternative Flows:

5b. Password fails to authenticate user. User is prompted to resubmit password.

26. Set Resource Utilization Defaults

Scope: SDR-1.

Level: User-Goal.

Primary Actor: Administrator.

Stakeholders and Interests:

- Administrators need the ability to set default numbers for utilization per person calculations, such as the amount of water needed per person, per day in a given environment.

Preconditions:

- Previously logged in.

Postconditions:

- System resource utilization default set.

Main Success Scenario:

1. User selects option to set default quantities.

2. System displays default list.

3. User submits default quantities.

4. System prompts user to confirm default with password.

5. User submits password.

6. System sets and saves default quantities.

7. System captures date, time, and IP address of user.

Alternative Flows:

6b. Password fails to authenticate user. User is prompted to resubmit password.

APPENDIX B. SDR-1 ARCHITECTURE

SDR-1 consists of three elements beyond the application code itself: Google Services, the App Engine Platform and Google Datastore. In the cloud environment, the exact deployment of these packages cannot be traced to a single server or set of servers, so they are depicted in Figure 18 as existing on servers within Google's cloud and referenced by their respective addresses. As discussed in Chapter IV, instances of applications are started on applications to handle incoming requests. In the case of SDR-1, the controller and part of the view will be instantiated on application servers, while static documents needed for the view will be served directly from optimized static files servers to increase performance. Model classes are persisted as entities in the Datastore where they can be retrieved by the controller.

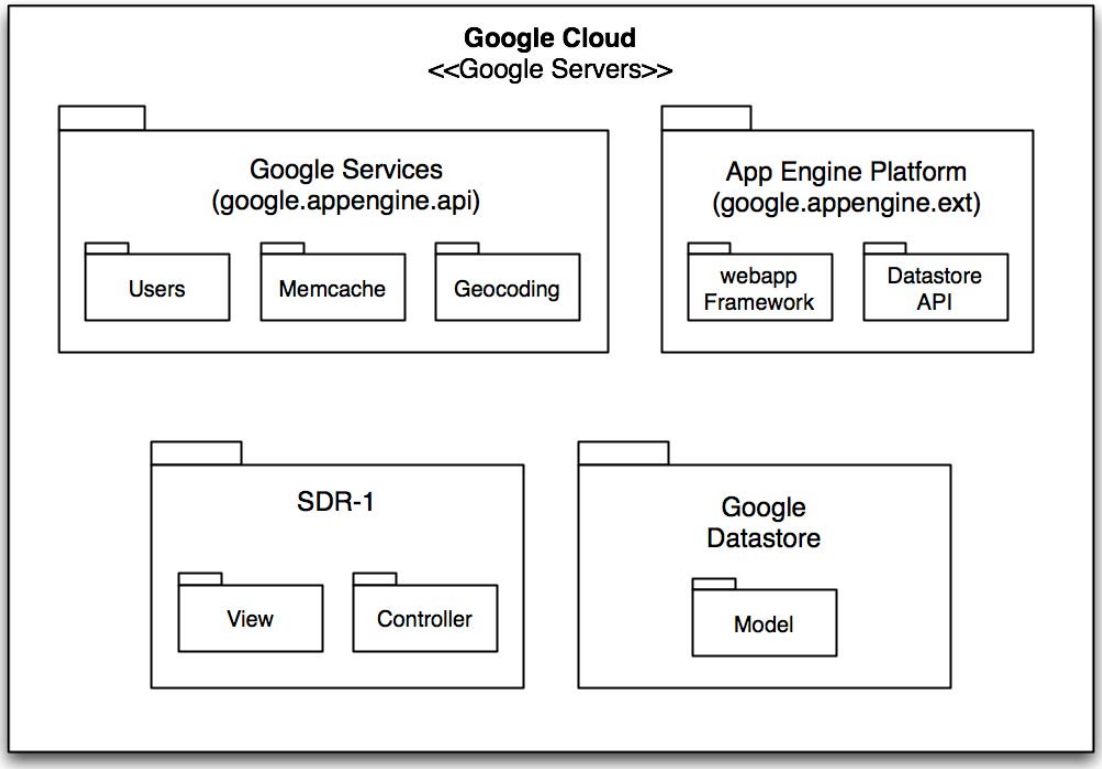


Figure 18. Architecture Overview

As described in Chapter IV, instances of applications are started on applications to handle incoming requests. In the case of SDR-1, the controller and part of the view will be instantiated on application servers, while static documents needed for the view will be served directly from optimized static files servers to increase performance. Model classes are persisted as entities in the Datastore where they can be retrieved by the controller.

Code comprising the view, Figure 19, is divided into two physical locations. Templates used to render pages based on data returned from the Datastore are deployed on application servers alongside main.py. All CSS and Javascripts should be deployed on static files servers to improve application performance.

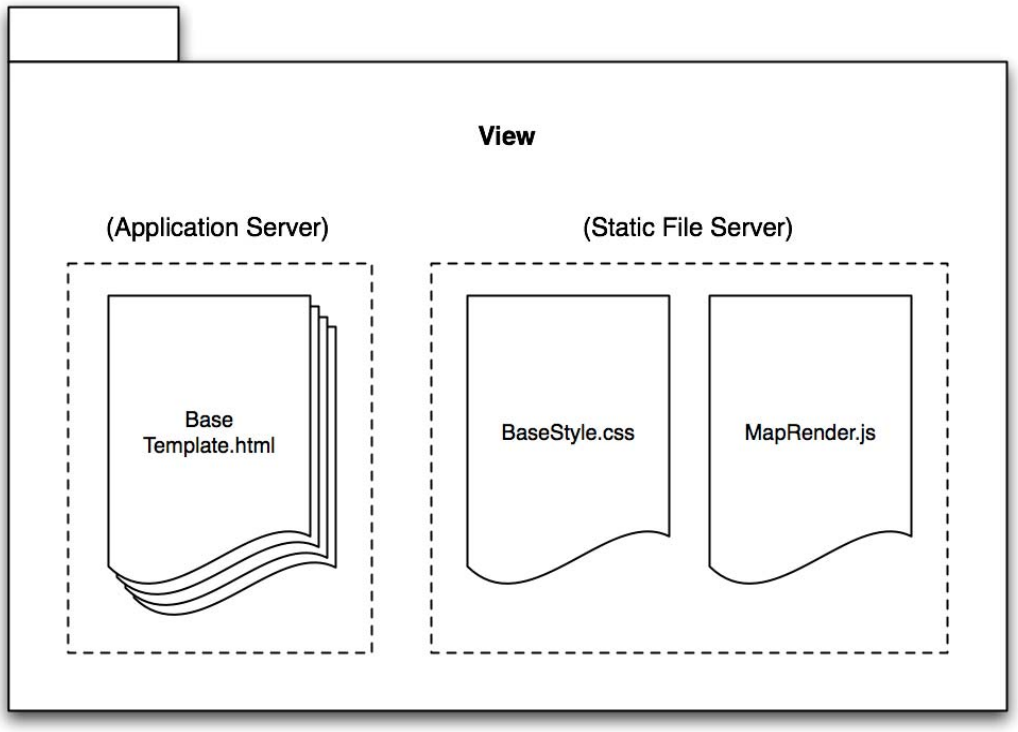


Figure 19. View Deployment

As described in Chapter IV, the controller, Figure 20, consists of two files: app.yaml and main.py. The app.yaml file is used by frontend servers to determine how incoming requests should be directed, while main.py contains the all of the application logic.

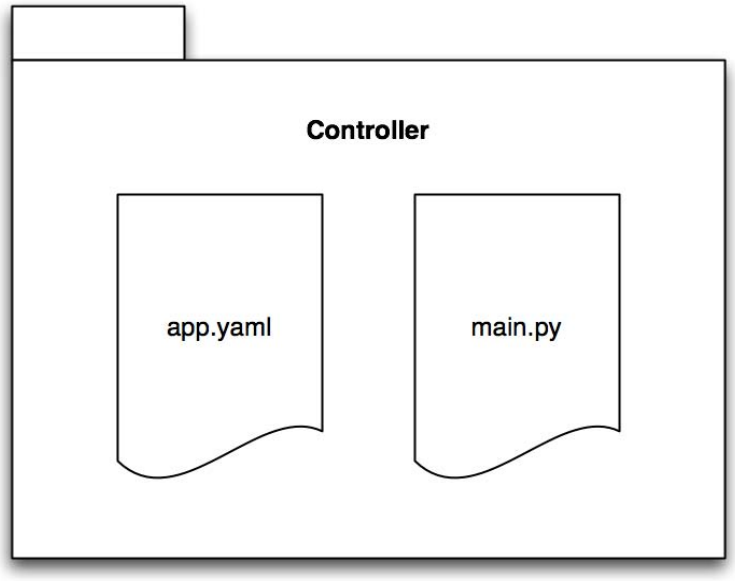


Figure 20. Controller Deployment

The main.py consists of several handlers to deal with incoming request as well as classes needed to exchange data with the Datastore. Deployment of these classes and locations of their superclasses, provided by Google App Engine, are illustrated in Figures 21 and 22.

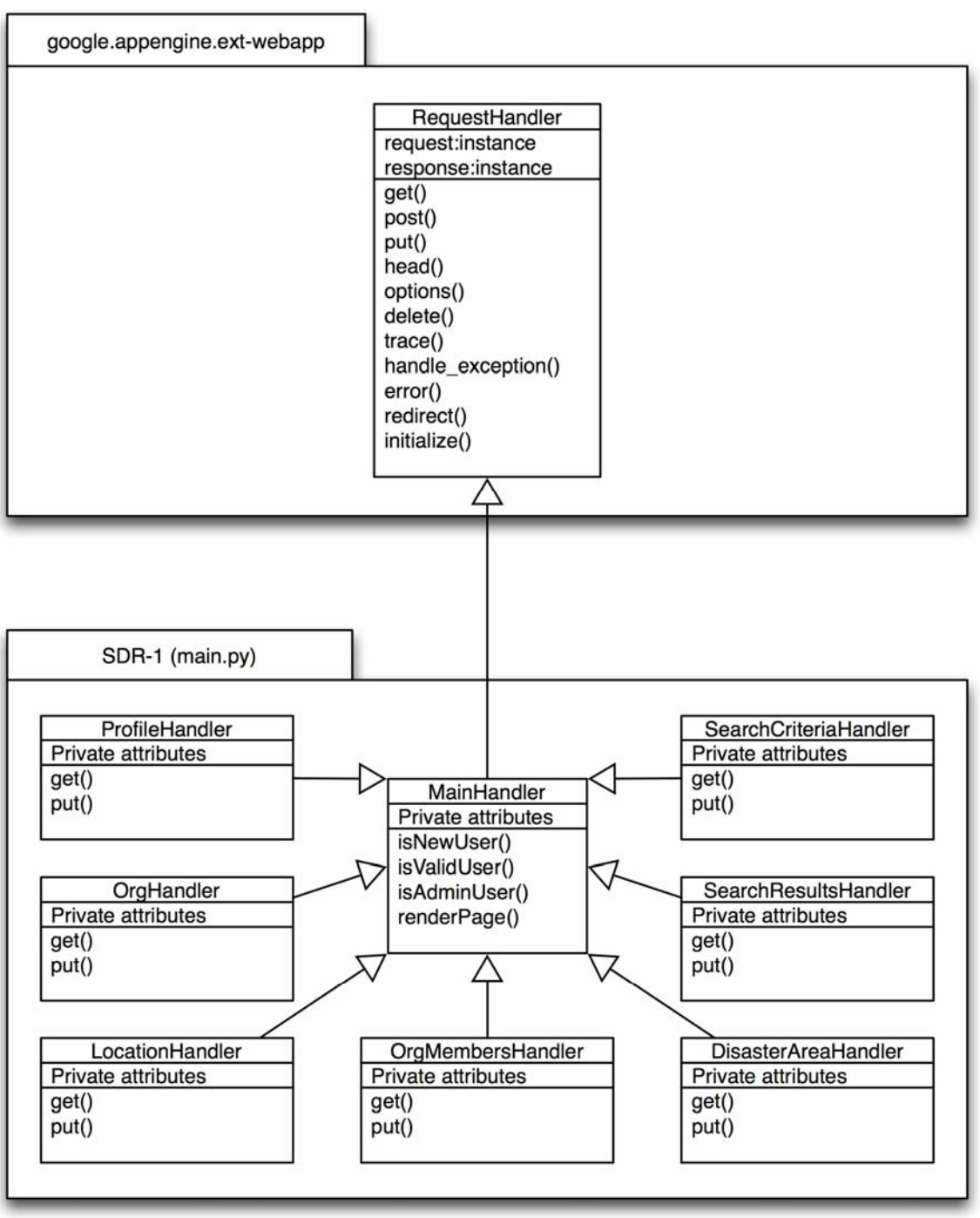


Figure 21. RequestHandler Classes and Subclass Deployment

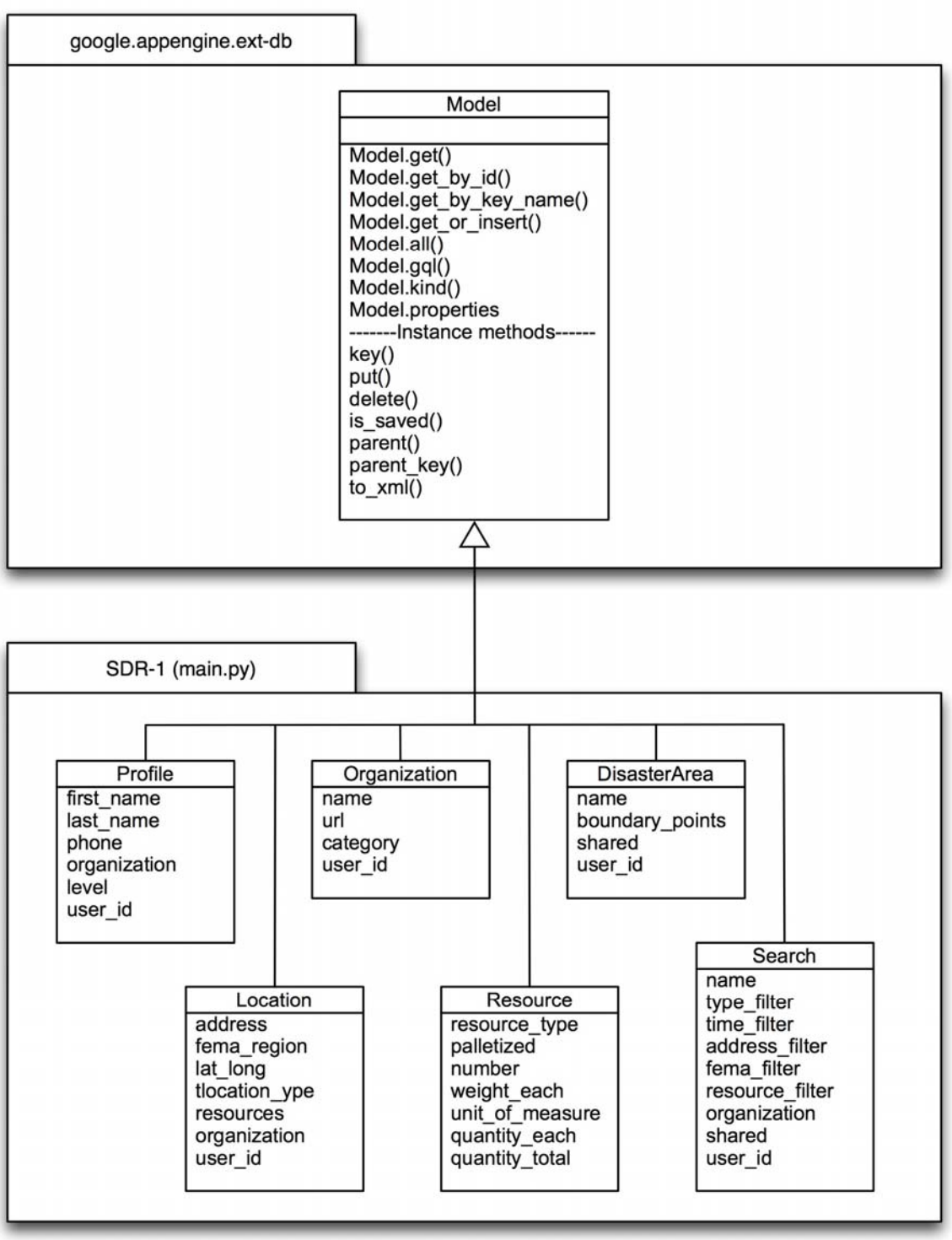


Figure 22. Model Classes and Subclass Deployment

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SDR-1 VIEWS

The Base template, Figure 23, defines the header, navigation, and footer for the entire application. All other templates within the application extend this template, allowing high-level changes to be made in only one place and cascaded throughout the application.

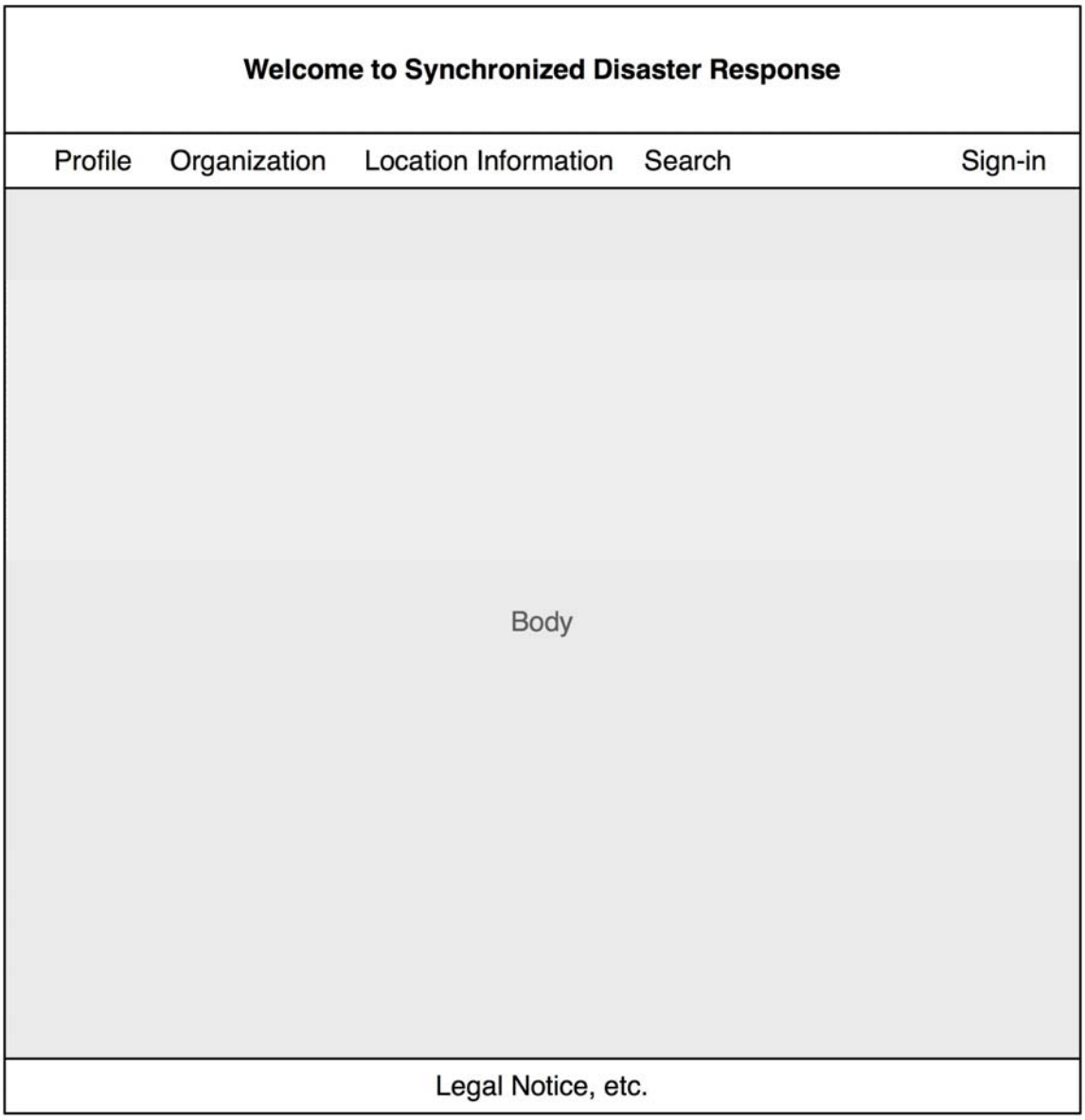


Figure 23. Base Template

The Add Profile template, Figure 24, is used to capture information about the user, including first and last name, e-mail address, and phone number. No username or password is required here since users authenticate with their Google or OpenID credentials. Information submitted here is tied to their Google or OpenID account.

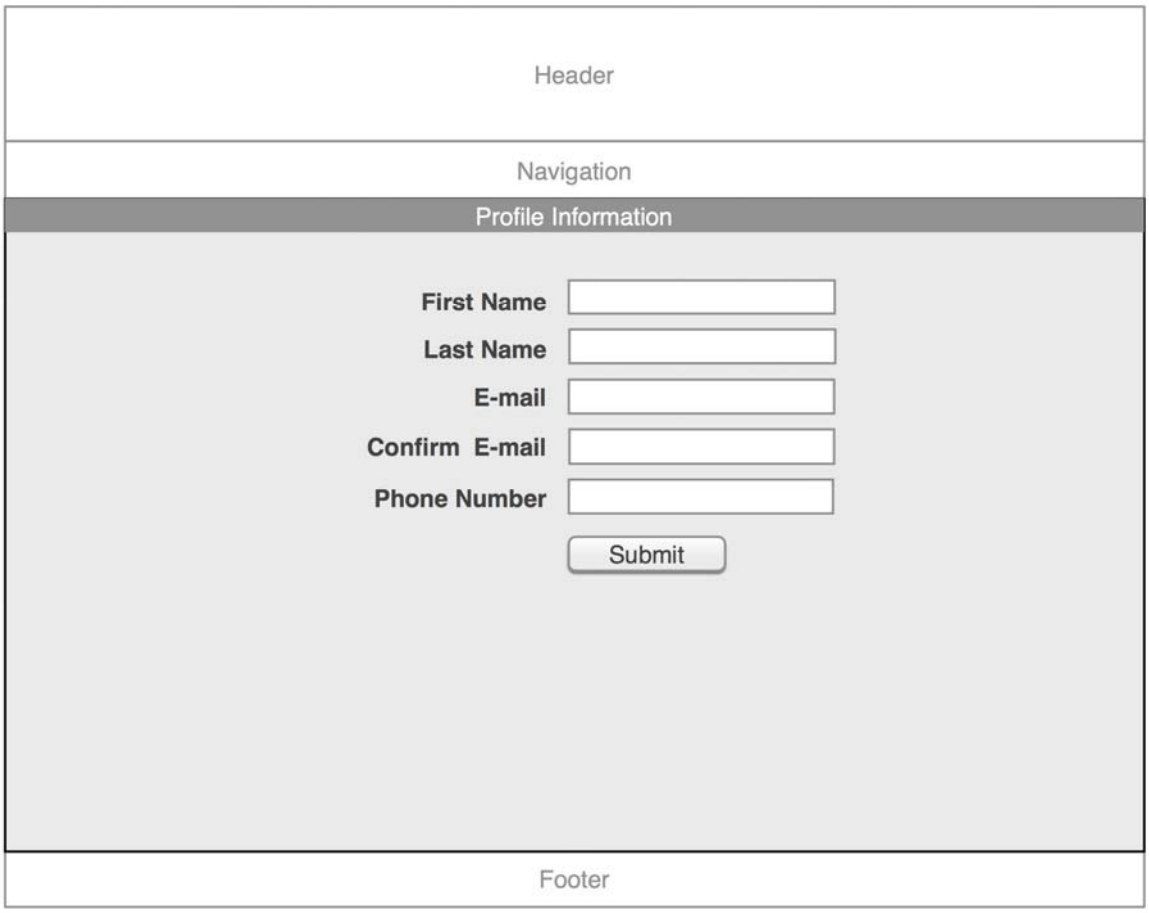


Figure 24. Add Profile Template

The Edit / Delete Profile template, Figure 25, is essentially the same as the Add Profile template. Previously submitted information is populated into the form for the user to edit. The permission level of each user is displayed in the profile, but it is not editable. When the user presses the update button, profile data updates. A user can also choose to delete their profile at anytime.

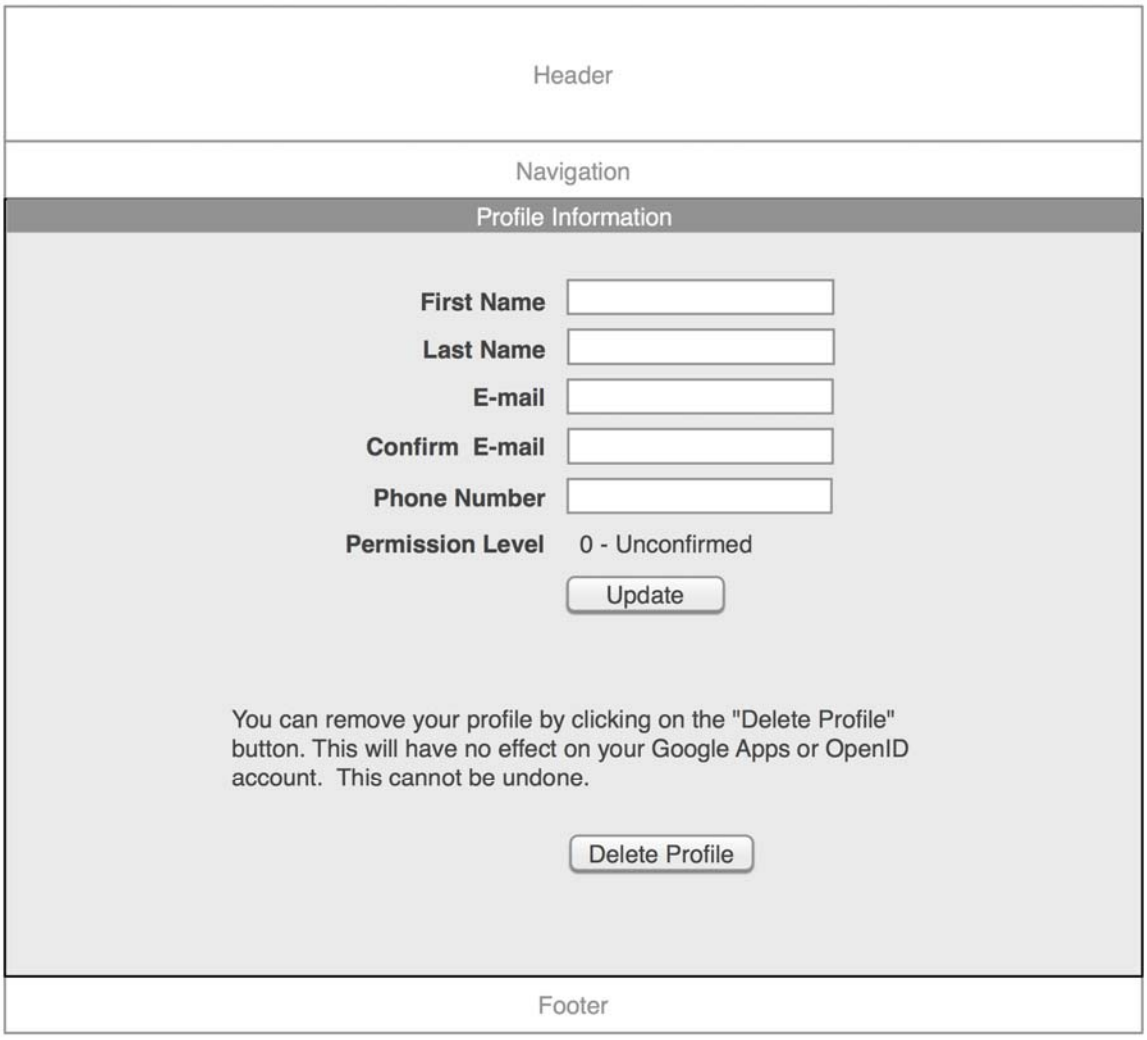


Figure 25. Edit / Delete Profile Template

The Add Organization template, Figure 26, is used to capture high-level information about an organization including its name, URL, and type. It is possible that organizations in different states will have the same name, so URL is used to differentiate organizations. An organization might have a headquarters and several stores. These are handled through the Locations template.

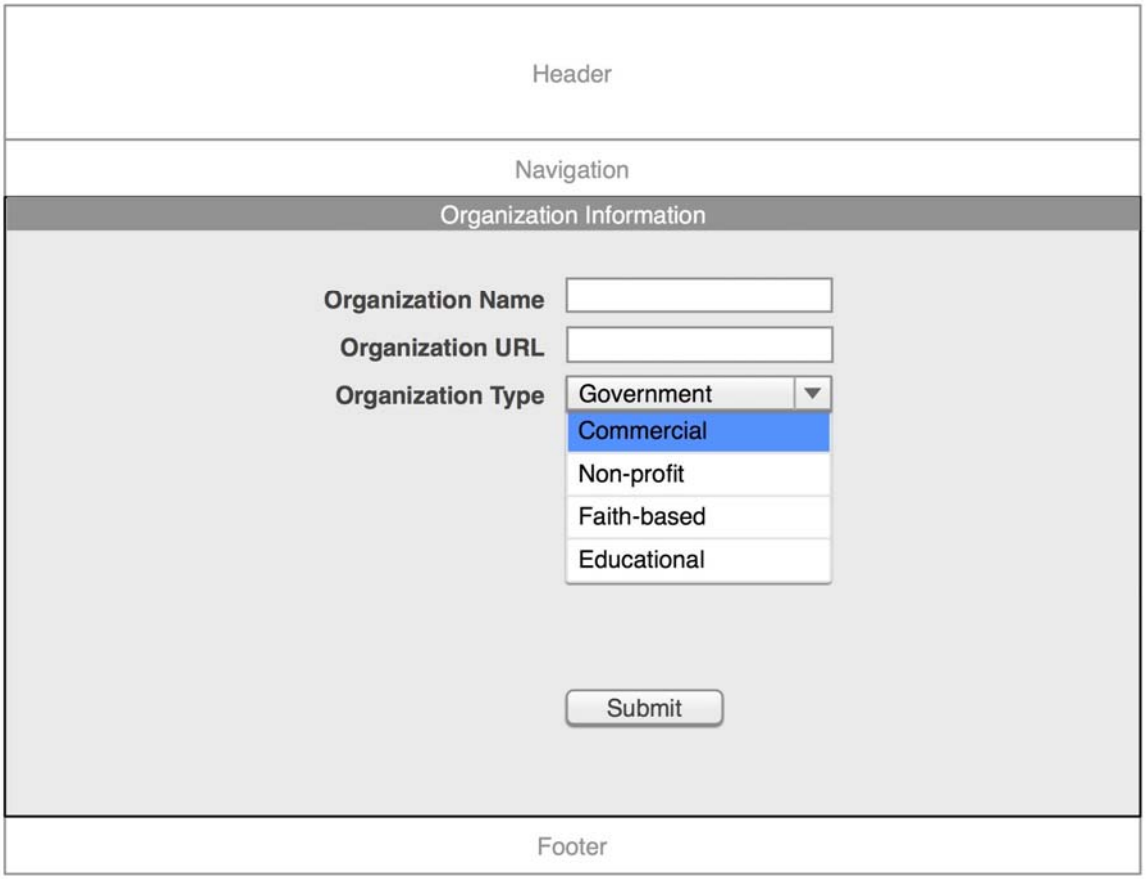


Figure 26. Add Organization Template

The Edit / Delete Organization template, Figure 27, is used to update organizational information. The name, URL, and type can be changed at any time and administrators have the ability to remove the organization.

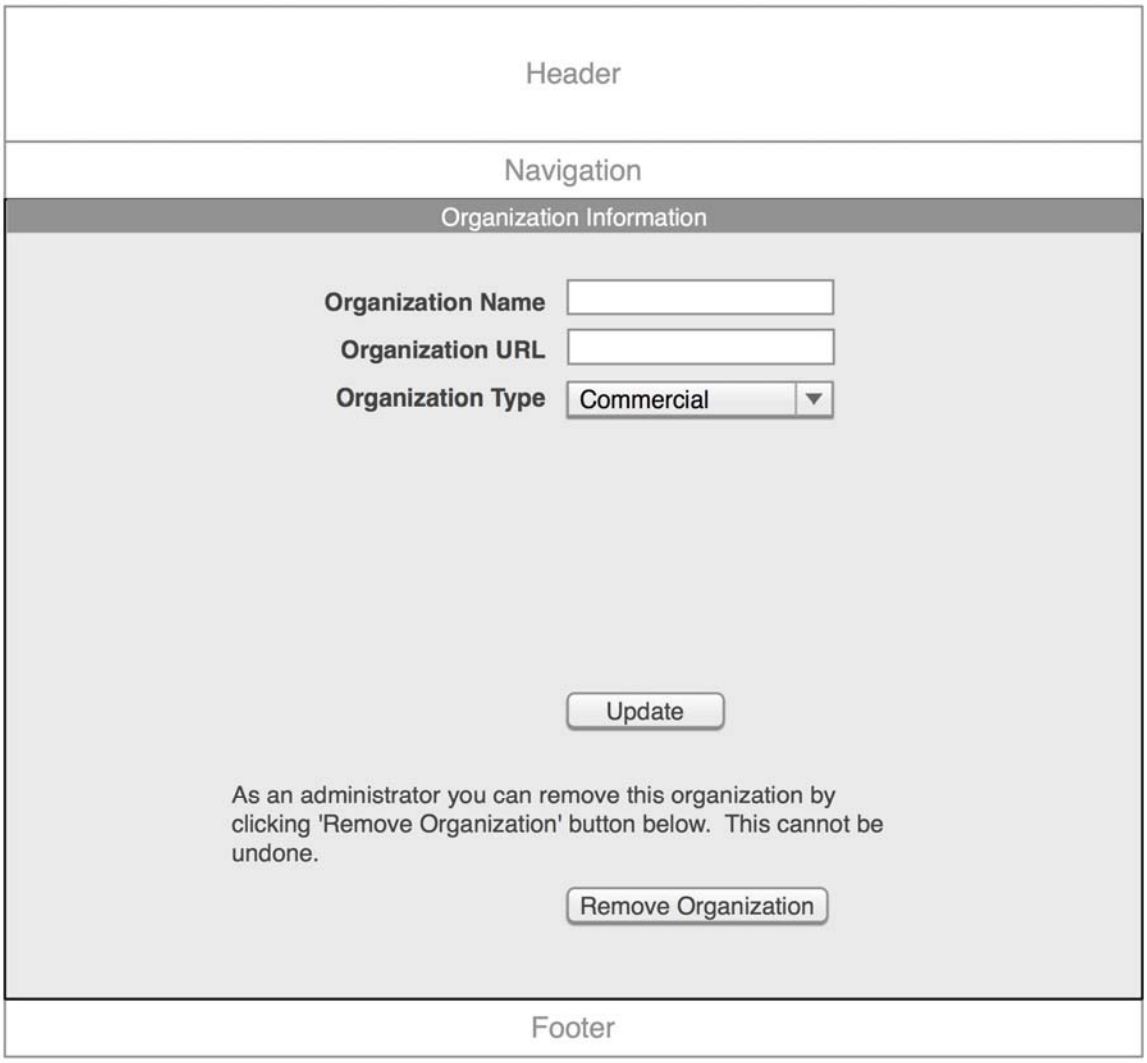


Figure 27. Edit / Delete Organization Template

The Set Permissions template, Figure 28, is used to set permission levels, ranging from Level 0 to Level 3, for users in an organization. This is only available to users with Level 3 permissions.

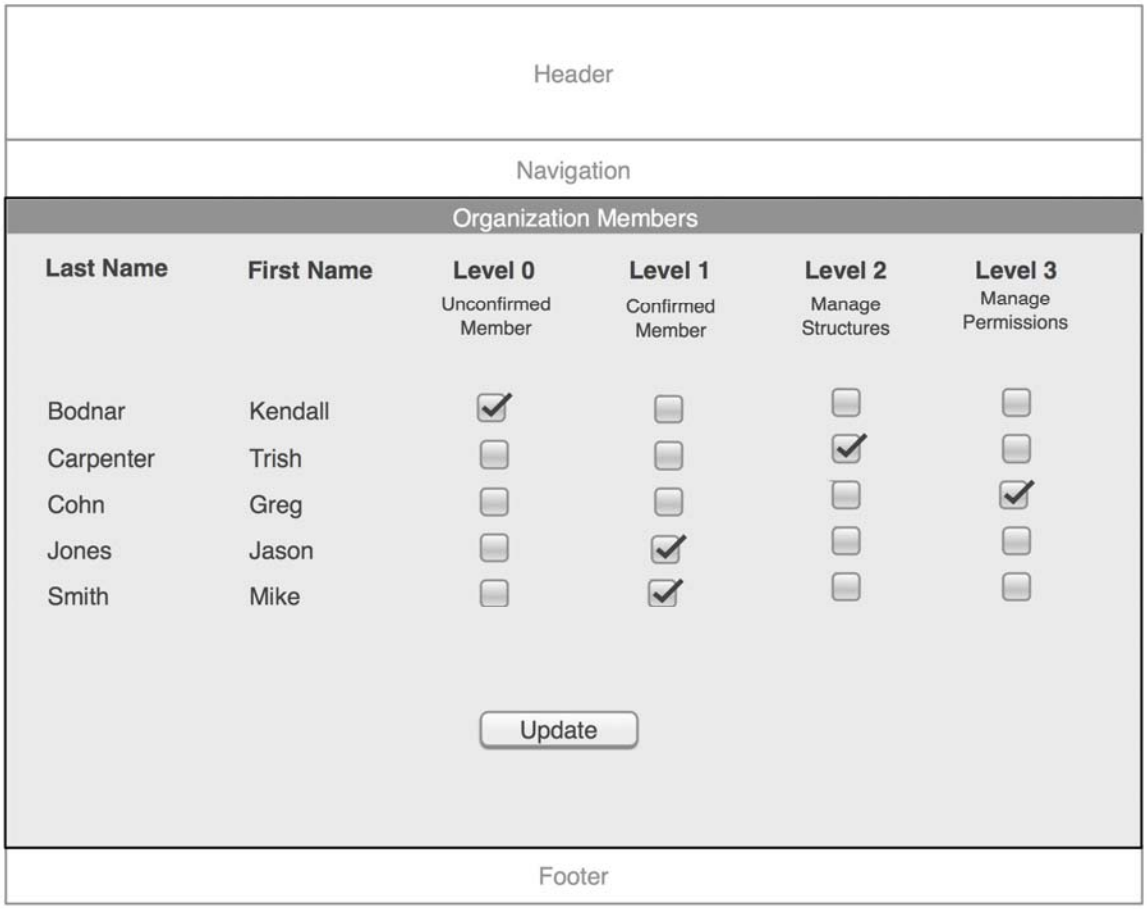


Figure 28. Set Permissions Template

The Add / Edit Location template, Figure 29, is used to capture all information about geographic positions and their associated resources. The position can be submitted as an address or lat-long coordinate, either of which will be used to resolve the other. There are currently seven types of locations: headquarters, warehouses, stores, requests, deliveries, and resources in transit. The first four represent physical structures with only one being allowed to occupy a particular geographic position. The last three are used to describe the evolving situation in a disaster area and could be stacked on the same geographic position over time. For example, one responder could submit a request for water at a specific position at 1200 and another responder could submit a different request at 1800. Both requests are allowed by the system, will be returned in a search, and will be included in any calculations.

Users have the option of submitting information in terms of pallets or raw numbers. Most organizations move and store resources using pallets so given the option to submit information in this format should reduce the initial burden. In all cases the total quantity is stored in non-palletized units, liters for example.

Recent submissions are listed at the bottom. This provides the user with ability to confirm an entry was saved, quickly adjust mistakes, or delete an entry from the system. If a user needs a list of all entries this can be accomplished with a search.

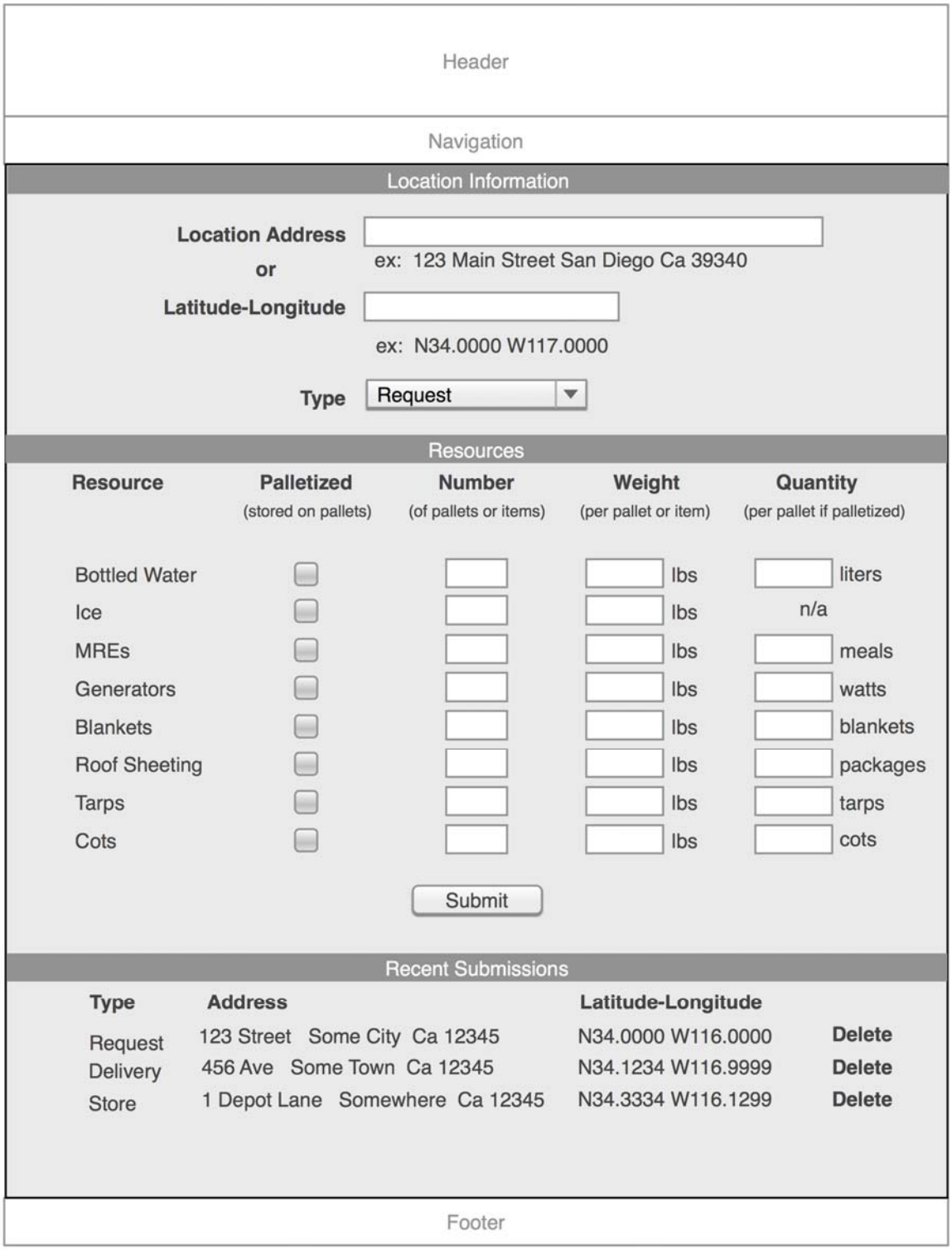


Figure 29. Add / Edit Location Template

The Define Search template, Figure 30, can be used to set detailed criteria for a search once and saved as a "Search" to be reused over time. The search will only be saved if a name is submitted in the Search Name field. Once saved, the search will show up in a list at the bottom of the page along with previously defined Searches. Executing one of the searches is accomplished by clicking on a search name. Searches can be edited or deleted as required.

By default every location in the system submitted in the past 48 hours will be returned. However, several options exist that allow data over a 96-hour period to be manipulated. Searches can be narrowed by organization, geographic area, FEMA region, time, location type, or resource. Searches can also be shared across an organization in which case they will appear with saved searches at the bottom of the page.

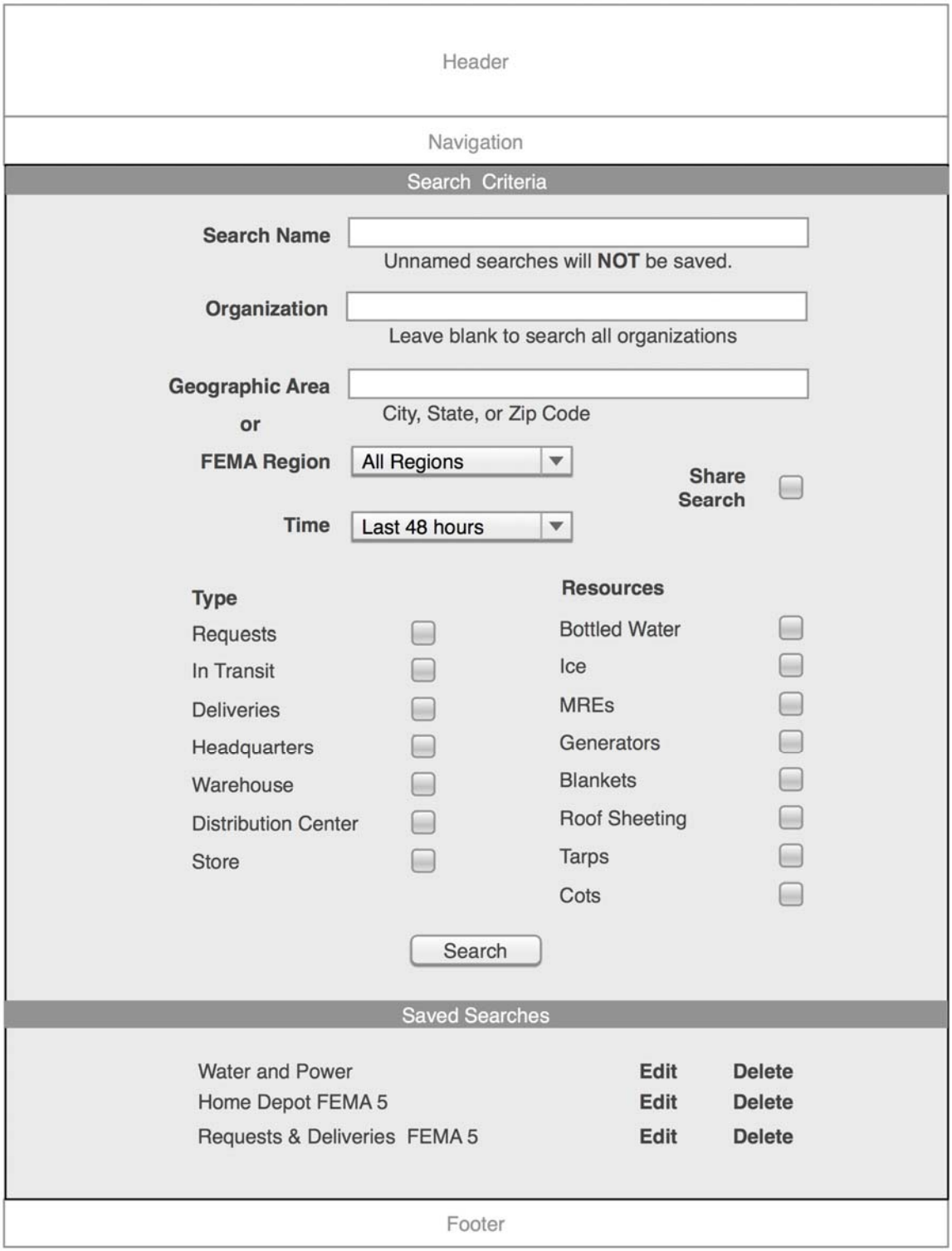


Figure 30. Define Search Template

The Search Results template, Figure 31, will display the results of a search on a map and as a list sorted by resource type. The advantage of sorting on resource type is that net quantities can be calculated based on the resource deliveries and requests and be treated as positive and negative numbers respectively. Resources in transit are treated as positive numbers as well but depicted as yellow dots on the map.

Two buttons are provided at the top of the page. Update executes the current search again, obtaining the most recent results. E-mail KML will execute a function that translates the locations into a KML and e-mails to the signed-in user.

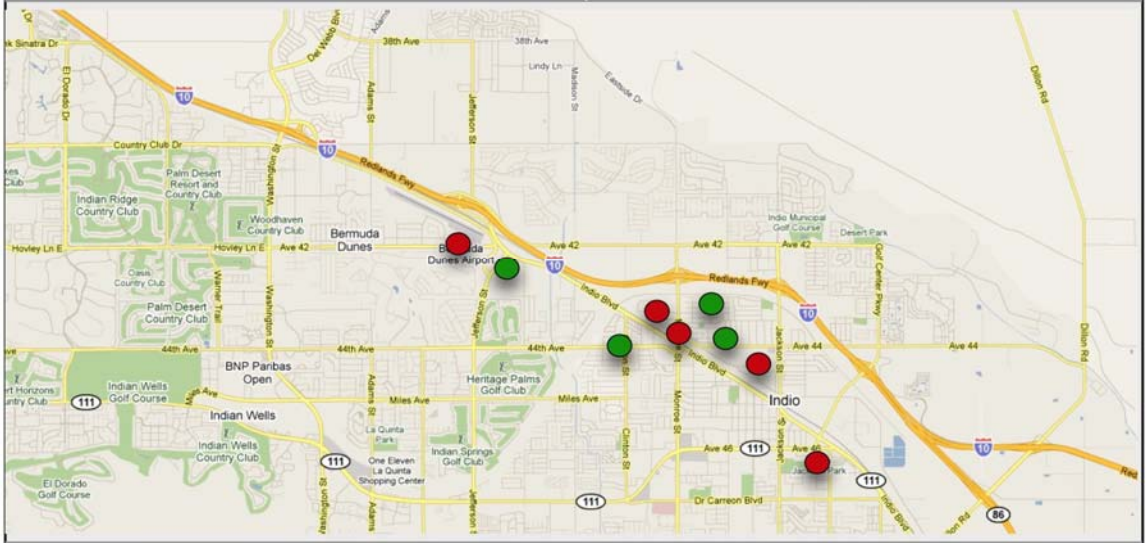
Header

Navigation

Search Results

Search Name: Water and Power Update E-mail KML

Map



Water

Type	Address	Latitude-Longitude	Quantity
Request	123 Street Some City Ca 12345	N34.0000 W116.0000	3000 liters
Delivery	456 Ave Some Town Ca 12345	N34.1234 W116.9999	4000 liters
Request	Unknown	N34.3334 W116.1299	2000 liters

Net Amount -1000 liters

Generators

Type	Address	Latitude-Longitude	Quantity
Request	123 Street Some City Ca 12345	N34.0000 W116.0000	1000 watts
Delivery	456 Ave Some Town Ca 12345	N34.1234 W116.9999	1250 watts

Net Amount 250 watts

Footer

Figure 31. Search Results Template

APPENDIX D. GOOGLE DOCUMENTATION

Additional information describing the Google App Engine Platform can be found on the Google App Engine Web site: http://code.google.com/appengine/docs/. Complete documentation of the Users Python API, Datastore API, Web application Framework, and services are provided there. Since the App Engine Platform is continuously evolving, a copy of the current documentation is provided with this thesis as a snapshot of what was used for designing SDR-1.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. USNORTHCOM FEEDBACK

10 Sep 10

MEMORANDUM FOR MAJOR SHAWN M. KELLY, USMC

Subject: Naval Post-graduate School thesis observations and comments.

1. Disclaimer. The positions taken in this memorandum are based on personal experience as a member of a NORAD and USNORTHCOM working group that that was developing solutions for an open-source, user-defined common operational picture to enhance disaster response situational awareness. The positions taken in this document do not reflect the official positions of NORAD and USNORTHCOM.

2. General Summary. Conceptually, the thesis answers all the most pertinent questions in order to code and implement a prototype that would have immediate impact and usability. The system design seems solid, balancing ease-of-use and simplicity, while remaining powerful for its purpose and intent. The cloud computing aspect keeps the system always available and scalable while alleviating issues that may arise concerning single organization ownership.

3. Detailed discussion of specifics.
a. The cloud computing concept is an excellent solution for this type of system. Not only will it potentially keeps costs low, but keeps ownership neutral, which is a key element of the system. It allows the community to be the owner, versus a particular organization, which reduces the potential of the system becoming proprietary or one entity controlling the content. Additionally, the fact that disasters can take out all communications in the affected area (which would include the ability to use SDR-1), cloud computing keeps the system always available. Replicating the system across working servers alleviates the problem that a centralized server setup presents if a disaster were to strike its physical location and render it unavailable.

b. The inclusion of a location Class that was not just defined by physical structures, but also includes the request, delivery and transit tags is a key addition to the system. This setup allows the user to see exactly what the picture looks like for a specific disaster at a specific time and specific location.

c. The palletized concept is a thoughtful and necessary inclusion for system implementation. This allows a user to understand if there is a requirement for additional personnel, forklifts and/or vehicles to load or off-load resources at the source or destination.

d. The ability for a saved search to be reused to re-query the system on latest information based on the original criteria set is extremely helpful. This concept simplifies Web-based searching and saves time for users.

e. Particularly liked the fact that search results are not stored, only the criteria used to define the search is. It seems that that this technique should allow the system to stay as

small and efficient as possible, but will still provide a way to do forensics on what searches have occurred, if the need arises.

f. The simplicity of the html structure and user interface was perfect. The way the html interface was designed keeps the output complex while proving the user an easy to use tool. Understanding that not all participants may be computer savvy, simple Web pages that deliver robust information is the target, and the concept, as designed, has taken that factor into account.

4. Detailed discussion/questions of concepts to be included.
a. For future iterations of the system, there may be need to expand FEMA's Big-8 resources to include medical supplies, and other large categories of resources that are commonly used in disaster response. Additionally, a requirement for a write-in text field would allow individuals to enter in information not listed as one of the standard resources. While it is understood that it may be difficult to document and search an unrestricted text field, repetitive data (resources not listed as a standard resource, but are being utilized over and over again as listed in the write-in field) can be pulled from the system and incorporated in later iterations as a standard resource offering the best solution to the end user.

b. While the ability to search by time is very useful, it may be beneficial to increase the periods offered in order to make the search more comprehensive based on response actions in relation to the disaster. Based on the National Response Framework, initial responders are on scene in the first 24 hours, state and National Guard responders between the 24-72 hour mark, and Federal/DOD responders after 72 hours. Understanding the needs of the disaster area following an incident, more hourly choices should be available, especially by front loading the initial response. As an example, choices at hours 2, 4, 8, 12, 24, 48, 72, 96, and 120 with the range increasing as the time from incident increases.

c. The search criteria currently return results by category only. While this is useful, results should be returned by either category or by most recent, as selected by the user. Very large organizations that transcend one resource, may want to know what resources across the spectrum have been requested most recently in order to fill that request as soon as possible.

d. An auto-update feature for search requests may be useful as well. This may be for the latest search request only, or as selected by the user if there are more than one. The interval of automatic updates may have to depend on the amount of data that has to be passed, or the how often data is entered in to the system, in order to keep it useful and efficient.

e. For future iterations, a way to view forensics or historical data for resource response to a particular incident would be very beneficial. This would be a very useful tool to model future disaster responses according to past successes and/or failures. Theoretically, this system would allow a user to see all of the activities for a particular disaster after the event. This information could be invaluable to major responder organizations such as FEMA as well as responders at all levels. Anyone could query historical data and make plans to enhance future response efforts after viewing lessons learned.

f. For additional security in future systems, a requirement to do forensics on who accesses the system and what data they are sharing and or requesting may be useful. This function may be currently available based on the log-in scheme that is discussed in the thesis.

g. For community-wide access, is it possible for someone to be added to the system if they are not attached to an organization? It may be important in the future to supply a way to track small-business, 'mom-and-pop' shops or even individual contributors to a disaster using this system. Not all of these will have a specific URL to make a unique ID in the system.

h. As the system develops, will there be a way to make push out an automatic reminder for personal or organizations to update their information? Since each URL entered gets a unique identifier in the system, there may be a need to figure out a way to resolve URLs if one changes, or if businesses change location, name, or resources.

i. The search function may need to have the ability to search using wild cards and/or Boolean functions, especially if the Resource Class contains a text-field. Understanding that not everyone will enter the similar information in the same way, this could be a challenge, but at least allows the community of users a more robust solution to query the disaster environments' needs and actions.

j. It seems that permissions may require some more fidelity. Understand that Google would control level 4 permissions inherent to their system, but which community of users manages original permissions, or manages or revokes level 3 permissions? The current design does not clearly address how permissions work. Prior to implementation, a more in depth look at how permissions are utilized, managed, and controlled may be required.

k. A final thought is to have some capability for phone-in requests. Responders may not have access to the system or resources in the disaster area to make the system work – under the current model, anyone can input information in to the system in the same way using an html template – but there may need to be discussion about a centralized call-in center (FEMA Regional Response Coordination Center, Joint Information Center or a Joint Field Office, etc). Because the system uses html interface, implementing this may not be out of the realm of possibilities.

5. Conclusion. Small questions regarding second-order aspects may help to solidify a more robust system if they are capable of being incorporated. As it is presented, the system is ready for implementation based on the thesis' design, concepts, and detailed structure. This is a thoroughly researched and well written thesis that needs a prototype built on its foundation soonest so that the program can move forward after a hand-on test.

ROGER A. SMITH
Major, USMC

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

About Google App Eng. (n.d.). Retrieved August 6, 2010 from http://www.google.com/support/a/bin/answer.py?hl=en&answer=91077

Amazon Web Services. (2010). Overview of Amazon Web Services, 9, 1–9.

American Logistics Aid Network Web Portal. (2010). Retrieved September 16, 2010 from http://www.alanaid.org/index.php

Bert, B. Celik, T., Hickson, I., & Lie, H. (2009). Cascading style sheets level 2 revision 1 (CSS 2.1) specification. Retrieved September 16, 2010, from http://www.w3.org/TR/CSS2/

Booth, D. et al. (2004). Web service architecture. W3C Working Group Note 11 February 2004. Retrieved August 24, 2010, from http://www.w3.org/TR/ws-arch/

Chang, F. et al. (2006). Bigtable: A distributed storage system for structured data. Retrieved August 5, 2010, from http://www.eecs.berkeley.edu/~culler/cs262b/summary/bigtable.html

Chu-Carroll, M. (2010). Code in the cloud: Programming Google App Engine. Pragmatic Programmers, LLC.

Fenton, R. (2007, February 20). DHS FEMA Region IX briefing for National Wildlife Suppression Association. Retrieved June 13, 2010, from http://www.nwsastraining.com/images/pdfuploads/FEMA%20Brief%20for%20NWSA.pdf

Google App Engine. (n.d.). Retrieved September 17, 2010, from http://code.google.com/appengine/docs/

Google Data Center Video. (2009). Retrieved August 5, 2010, from http://blogoscoped.com/archive/2009-04-08-n39.html

Heide, E. (1989). Disaster response: Principles of preparation and coordination. Retrieved August 6, 2010, from http://orgmail2.coe-dmha.org/dr/flash.htm

Logimax Web site. (n.d.). Retrieved September 16, 2010, from http://www.e-logimax.com/wms-solutions/wms-foundation.php

Sanderson, D. (2010). Programming Google App Engine. Retrieved September 16, 2010, from http://oreilly.com/catalog/9780596522735/preview#preview

SCB Software Web site. (n.d.). Retrieved September 15, 2010, from http://home.comcast.net/~scbonanno/

Severance, C. (2009). *Using Google App Engine: Building Web applications.* Sebastopol, CA: O'Reilly Media

Sun Microystems, Inc. White Paper. (2009) Introduction to Cloud Computing Architecture. Retrieved June 13, 2010, from http://webobjects.cdw.com/webobjects/media/pdf/Sun_CloudComputing.pdf

Thibeau, D., & Reed, D. (2009, August 10). Open trust framework for open government: Enabling citizens involvement through open identity technology. *OpenID Foundation and Information Card Foundation*, 10, 1–10.

Thomas, Anisya (n.d.). Humanitarian logistics: Enabling disaster response. Fritz Institute. Retrieved June 13, 2010, from http://www.fritzinstitute.org/PDFs/WhitePaper/EnablingDisasterResponse.pdf

Wassenhove, Van. (2006). Blackett Memorial Lecture Humanitarian Aid Logistics: Supply Chain Management in High Gear. Retrieved June 13, 2010, from http://errrmsystems.pbworks.com/f/vanwassenhove.pdf

What is GIS?. (n.d.). Retrieved September 16, 2010, from http://www.gis.com/content/what-gis

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Peter Denning
   Naval Postgraduate School
   Monterey, California

4. Professor Dan Boger
   Naval Postgraduate School
   Monterey, California

5. Lt. Col. Karl Pfeiffer, USAF
   Naval Postgraduate School
   Monterey, California

6. Professor Man-Tak Shing
   Naval Postgraduate School
   Monterey, California

7. Mr. John Shea
   Office of the DoD CIO
   Arlington, Virginia

8. COL Kevin Foster, USA
   Office of the DoD CIO
   Arlington, Virginia

9. Professor Bret Michael
   Naval Postgraduate School
   Monterey, California

10. Professor George Dinolt
    Naval Postgraduate School
    Monterey, California

11. Professor Doron Drusinsky
    Naval Postgraduate School
    Monterey, California

12. Professor Thomas Otani
Naval Postgraduate School
Monterey, California

13. Professor Loren Peitso
Naval Postgraduate School
Monterey, California

14. Mr. Alex Nelson
Naval Postgraduate School
Monterey, California

15. Mr. Scott J Dowell
Computer Science Corporation
San Diego, California

16. Mr. Michael Lee
Touchstone Consulting Group
Washington, D.C.

17. Ms. Karen Gordon
Institute for Defense Analyses
Alexandria, Virginia

18. Marine Corps Representative
Naval Postgraduate School
Monterey, California

19. Director, Training and Education, MCCDC, Code C46
Quantico, Virginia

20. Director, Marine Corps Research Center, MCCDC, Code C40RC
Quantico, Virginia

21. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
Camp Pendleton, California